DML – VIEWS, FUNCTIONS E PROCEDURES

ANTONIO DAVID VINISKI
antonio.david@pucpr.br
PUCPR

VIEWS

O QUE SÃO VIEWS?

- Uma View é um objeto que pertence a um banco de dados.
  - Sua definição é baseada em declarações **SELECT**'s.
  - Retorna uma determinada visualização de dados de uma ou mais tabelas.
- Esses objetos são chamados de "virtual tables".
  - não fazem parte do esquema físico da base.
  - Formada a partir de outras tabelas que por sua vez são chamadas de "based tables" ou ainda outras Views.

BENEFÍCIOS DAS VIEWS

- Uma forma de salvar consultas no banco de dados.
- Pode ser utilizada para promover restrições em dados para aumentar a segurança dos mesmos e definir políticas de acesso em nível de tabela e coluna.
  - Podem ser configurados para mostrar colunas diferentes para diferentes usuários do banco de dados;
- Pode ser utilizada com um conjunto de tabelas.
- Suportam as cláusulas **JOIN** e **UNION**.

CRIANDO VIEWs

CREATE [**OR REPLACE**] [**ALGORITHM** = algorithm_type] **VIEW** view_name [(column_list)]

AS select_statement;

- **OR REPLACE**: pode ser utilizada para substituir uma View de mesmo nome existente no banco de dados ao qual ela pertence. Pode-se utilizar ALTER TABLE para o mesmo efeito;
- **ALGORITHM**: essa cláusula define qual algoritmo interno utilizar para processar a View quando a mesma for invocada. Estes podem ser UNDEFINED (cláusula em branco), MERGE ou TEMPTABLE.

VIEW - EXEMPLO

- Crie uma view que retorne as informações dos clientes do restaurante (id, nome, data_nasc, data_criacao) e também o nome e a quantidade de cada um dos produtos consumidos por eles.
  - As informações de cliente estão disponíveis nas tabelas pessoa e cliente.
  - Para acessar as informações de produto, precisa-se identificar as comandas de cada cliente, os registros de produtos de cada comanda e, por fim, as informações dos produtos consumidos (Junção com as tabelas comanda, registro e produto).
  - Para calcular a quantidade de cada um dos produtos deve-se utilizar função de agregação.

VIEW - EXEMPLO

```
CREATE OR REPLACE VIEW verifica_produtos_clientes (id_cliente, nome_cliente, data_nasc, data_criacao,
                                                   produto, count_registro, quantidade)
    AS
    SELECT c.id, p.nome, p.data_nasc, c.data_criacao,
                pr.nome AS "Produto",
                COUNT(r.id_produto) AS "Número de Registro",
                SUM(r.quantidade) AS "Quantidade Vendida"
    FROM pessoa AS p
    INNER JOIN cliente AS c ON c.id = p.id
    INNER JOIN comanda AS co ON co.id_cliente = c.id
    INNER JOIN registro AS r ON r.id_comanda = co.id
    INNER JOIN produto AS pr ON pr.id = r.id_produto
    GROUP BY p.id, c.id, pr.nome
    ORDER BY c.id;
```

EXERCÍCIOS I

PROCEDURES

Criar uma **VIEW** para:

1. Consultar o id, nome, data_nasc, data_criacao, número de comadas, valor total das comandas de cada um dos clientes.
2. Retornar o id, nome, data_nasc, telefone, email, valor_hora, número de registros, valor total registrado, quantidade de produtos registrados de cada um dos garçons.
3. Retornar a id, nome, data e o valor de cada um dos pagamentos dos garçons.
4. Retornar o id, data_criacao , id_cliente e todos os campos da tabela registro associada a de cada uma das comandas.

PROCEDURES

INTRODUÇÃO

PROCEDURES

- Quando desenvolvemos aplicações que acessam banco de dados (boa parte delas), é comum executarmos rotinas complexas de manipulação desses dados a partir da linguagem/ferramenta utilizada.
  - Para isso, utilizamos várias instruções SQL em sequência para obter o resultado esperado.
- Isso pode requerer várias consultas e atualizações na base, o que acarreta um maior consumo de recursos pela aplicação.
  - No caso de aplicações web, isso se torna ainda mais visível, devido a maior quantidade de informações que precisam trafegar pela rede e de requisições ao servidor.

Como melhorar o desempenho da aplicação?

MELHORANDO O DESEMPENHO

PROCEDURES

- Uma boa forma de contornar ou ao menos atenuar esse consumo de recursos diretamente pela aplicação é transferir parte do processamento direto para o banco de dados.
  - Considerando que as máquinas servidoras geralmente têm configurações de hardware mais robustas.
- Como executar várias ações no banco de dados a partir de uma única instrução?
  - ***Stored Procedures*** (ou Procedimentos Armazenados, em português).

O QUE SÃO STORED PROCEDURES?

PROCEDURES

- São rotinas definidas no banco de dados, identificadas por um nome pelo qual podem ser invocadas.
  - Um procedimento desses pode executar uma série de instruções, receber parâmetros e retornar valores.
- Podem ser usados para validação de dados, controle de acesso, execução de declarações SQL complexas e muitas outras situações.
- Suportam declaração de variáveis, verificação de valores por meio de condições ([**IF**, **ELSEIF**, **ELSE**], [**CASE WHEN THEN ELSE**]).
- Podem retornar valores e resultados de consultas.

PONTOS POSITIVOS E NEGATIVOS

PROCEDURES

- Pontos positivos:
  - Simplificação da execução de instruções SQL pela aplicação.
  - Transferência de parte da responsabilidade de processamento para o servidor.
  - Facilidade na manutenção, reduzindo a quantidade de alterações na aplicação.
- Pontos negativos:
  - Necessidade de maior conhecimento da sintaxe do banco de dados para escrita de rotinas em SQL;
  - As rotinas ficam mais facilmente acessíveis. Alguém que tenha acesso ao banco poderá visualizar e alterar o código.

CRIANDO UMA *STORED PROCEDURE*

PROCEDURES

Especifica o início da criação de uma procedure

Define o nome do procedimento, o qual será utilizado para invocá-lo posteriormente

Especifica os parâmetros que o procedimento recebe para realizar as operações necessárias.

```
DELIMITER $$
CREATE PROCEDURE nome_procedimento (parâmetros)
BEGIN
    /*CORPO DO PROCEDIMENTO*/
END $$
DELIMITER ;
```

Onde são definidas todas as instruções que serão executadas pelo procedimento armazenado

Indica que a criação do procedimento chegou ao fim.

Altera novamente o delimitador para que o ';' indique novamente o fim de uma instrução

PARÂMETROS DA STORED PROCEDURE

PROCEDURES

- Os "parâmetros" são opcionais e, caso não sejam necessários, devem permanecer apenas os parênteses vazios na declaração da ***procedure***. Para que um procedimento receba parâmetros, é necessário seguir certa sintaxe (dentro dos parênteses):
  - (**MODO** nome **TIPO**, **MODO** nome **TIPO**, **MODO** nome **TIPO**).
- O "nome" e o "**TIPO**" dos parâmetros segue as mesmas regras de definição de campos, variáveis. O "**MODO**" indica a forma como o parâmetro será tratado no procedimento:
  - **IN**: indica que o parâmetro é apenas para entrada/recebimento de dados;
  - **OUT**: usado para parâmetros de saída. Para esse tipo não pode ser informado um valor direto (como 'teste', 1 ou 2.3), deve ser passada uma variável "por referência";
  - **INOUT**: como é possível imaginar, este tipo de parâmetro pode ser usado para os dois fins (entrada e saída de dados). Nesse caso também deve ser informada uma variável e não um valor direto.

UTILIZAR - EXCLUIR UMA PROCEDURE

PROCEDURES

- Para utilizar um ***Stored Procedure*** criada utilizamos o comanda **CALL**.
  - **CALL** <nome_procedure>.
- Para excluir uma ***Stored Procedure*** utilizamos o comando **DROP**.
  - **DROP PROCEDURE IF EXISTS** <nome_procedure>.

Quando usar *Stored Procedures*?

PROCEDURES

Quando precisamos de múltiplas instruções para a realização de uma operação relacionada a aplicação, ou ainda para a realização de operações específicas que exijam um ou mais parâmetros.

- Exemplo: Criação de um novo cliente do restaurante.
- Instruções:
  - Criar um novo registro na tabela ***pessoa***.
  - Verificar o ID do registro na tabela ***pessoa***.
  - Inserir um novo registro na tabela ***cliente.***
  - Inserir um novo registro na tabela ***comanda.***
  - Retornar o **ID** da comanda criada para o novo cliente.
- A execução de cada uma dessas instruções separadamente gera várias requisições ao servidor, o que pode acarretar uma redução no desempenho dos mesmo.

PROCEDURE - EXEMPLO

```
DELIMITER $$
CREATE PROCEDURE IF NOT EXISTS inserindo_novo_cliente(IN nome VARCHAR(24), IN cpf
                                                      CHAR(11), IN sexo CHAR(1), IN data_nasc DATE)
BEGIN
    DECLARE pessoa_id, comanda_id INT DEFAULT NULL;
    IF nome IS NOT NULL AND cpf IS NOT NULL AND data_nasc IS NOT NULL THEN
        INSERT INTO restaurante.pessoa VALUES (NULL, nome, cpf, sexo, data_nasc);
        SET pessoa_id := (SELECT LAST_INSERT_ID());
        INSERT INTO restaurante.cliente VALUES (pessoa_id, NOW());
        INSERT INTO restaurante.comanda VALUES (NULL, NOW(), 0.0, 0, pessoa_id, NULL);
        SET comanda_id := (SELECT LAST_INSERT_ID());
    END IF;
    SELECT comanda_id;
END $$
DELIMITER ;
```

EXERCÍCIOS II

PROCEDURES

1. Criar uma ***Stored Procedure*** para selecionar o faturamento do restaurante por dia da semana em um ano específico (ano é passado como parâmetro).
2. Criar uma ***Stored Procedure*** para realizar o pagamento de um funcionário serviços gerais em um mês específico, caso ainda não tenha sido pago.
3. Criar uma ***Stored Procedure*** para inserir um novo funcionário garçom no banco de dados.
4. Criar uma ***Stored Procedure*** para inserir um novo funcionário serviços gerais no banco de dados.
5. Criar uma ***Stored Procedure*** para realizar o pagamento de um funcionário garçom, considerando todas as suas horas trabalhadas não pagas, bem como as comissões referente aos produtos registrados (recebe como argumento o ID do garçom).

EXERCÍCIOS III

PROCEDURES

4. Criar uma ***Stored Procedure*** para retornar o número de funcionários, número de funcionários garçons, número de funcionários serviços gerais e o número de clientes do banco de dados, recebendo como parâmetro um período específico (data_inicio, data_fim).
   A. Caso a data de início não seja estabelecida, retornar desde os primeiros registros, até a data_fim.
   B. Caso a data fim não seja informada, retornar os dados a partir da data_inicio.
   C. Caso nenhuma das datas seja informada, considerar todos os registros do banco de dados.
5. Criar uma ***Stored Procedure*** para inserir o registro de um produto em uma comanda, atualizar o valor total (valor_total) e o número de itens (num_items) da comanda e também diminua a quantidade do produto no estoque (**OBS**: os parâmetros da ***procedure*** são: valor_produto, quantidade, id_comanda, id_garcom, id_produto).

FUNCTIONS

INTRODUÇÃO

FUNCTIONS

- Assim como as ***procedures***, uma função é um tipo de rotina armazenada no banco de dados. Uma função é utilizada para gerar um valor que pode ser usado em outras instruções.
    - Esse valor é geralmente baseado em um ou mais parâmetros fornecidos à função.
- As funções são executadas geralmente como parte de uma expressão (chamadas em uma instrução **SELECT** por exemplo).
- O MySQL possui diversas funções internas que o desenvolvedor pode utilizar (como mostradas anteriormente: operações com *String* e Data), e também permite que criemos nossas próprias funções.
- Uma função é considerada "determinística" se ela sempre retorna o mesmo resultado para os mesmos parâmetros de entrada, e "não determinística" caso contrário.

CRIANDO UMA FUNÇÃO

FUNCTIONS

```
DELIMITER $$
CREATE FUNCTION nome_funcao (parâmetros)
RETURNS retorna_valor [NOT] DETERMINISTIC
BEGIN
    /* CORPO DA FUNÇÃO */
END $$
DELIMITER ;
```

Especifica os parâmetros que a função recebe. Diferente da ***Stored Procedure***, a função não especifica o **MODO** do parâmetro, somente o "nome" e o "**TIPO**"

Especifica o "**TIPO**" do retorno da função definida

UTILIZANDO UMA FUNÇÃO

FUNCTIONS

- As funções são utilizadas em conjunto com outras instruções.
  - **SELECT** nome_funcao(parâmetros);
  - **INSERT INTO** tabela **VALUES**(nome_funcao(parâmetros));
  - **UPDATE** tabela **SET** campo = (nome_funcao(parâmetros));

EXEMPLO FUNÇÃO

FUNCTIONS

- Criar um função para retornar o dia da semana (em português) dado um número inteiro passado como parâmetro.

PROCEDURE - EXEMPLO

```
DELIMITER $$
CREATE FUNCTION weektext(day_of_week INT)
RETURNS VARCHAR(30) DETERMINISTIC
BEGIN
        DECLARE tex_week_day VARCHAR(20);
        SET tex_week_day := (SELECT  CASE WHEN day_of_week = 1 THEN "Domingo"
                                          WHEN day_of_week = 2 THEN "Segunda"
                                          WHEN day_of_week = 3 THEN "Terça"
                                          WHEN day_of_week = 4 THEN "Quarta"
                                          WHEN day_of_week = 5 THEN "Quinta"
                                          WHEN day_of_week = 6 THEN "Sexta"
                                          ELSE "Sábado"
                                    END);
        RETURN  tex_week_day;
END$$
```

EXERCÍCIOS IV

PROCEDURES

Crie funções para:

1. Contar o número de comandas de um cliente.
2. Retornar o valor a receber de comissão de um garçom.
3. Retornar a quantidade vendida de um produto.
4. Retornar o faturamento total de um produto.
5. Retornar o número de registros efetuados por um garçom.