

PROCEDIMENTOS ARMAZENADOS

(STORED PROCEDURES)

Introdução

- No dia a dia do trabalho com banco de dados, surgem problemas relacionados à necessidade de realizar operações complexas que envolvem a realização de um ou mais comandos de **seleção** (*select*) em conjunto com comandos de **manipulação** (*update* e *delete*) de dados;
- Também é preciso realizar operações no banco de dados com base em **parâmetros** fornecidos pelo **usuário** do SGBD ou Aplicação;

Introdução

- Nessas situações, o ideal é que essas **operações não sejam definidas na aplicação** que vai fazer uso do banco de dados, **mas no próprio servidor do banco de dados;**
- Isso garante a segurança do banco de dados, uma vez que o mesmo pode **controlar o acesso a essas operações especiais**, evitando o conhecimento excessivo da estrutura do banco de dados pelos desenvolvedores de aplicações;

Introdução

- Além da questão de segurança do banco de dados, já que o usuário só poderá executar os procedimentos a que tiver acesso, os **procedimentos armazenados** ajudam a **diminuir o tráfego de informações entre o usuário e o servidor**, tornando mais simples o processamento das informações e melhorando o **desempenho** da aplicação;
- Os **Procedimentos Armazenados** são definidos como um conjunto de comandos da linguagem **SQL** definidos no **servidor do banco de dados** e **acionados por eventuais chamadas** de qualquer usuário que tenha autorização para sua execução;

Exemplo Prático

- ❑ Você faz parte de uma equipe de desenvolvimento de uma aplicação para o **Hotel Dorme Bem**. Suponhamos que a camada de **Aplicação** deseja fazer uma consulta que retorna a **lista de produtos disponíveis** em um determinado **quarto** de hotel;
- ❑ Esse tipo de consulta é muito comum e são chamados de **Relatórios Gerenciais** nos sistemas comerciais;
- ❑ Você então recebe a tarefa na Sprint de programar esse relatório e para consultar o banco de dados escreve o seguinte código:

Consulta

- ❑ **SELECT**
- ❑ **quarto.número_qua as 'Número do Quarto',**
- ❑ **produto.descrição_prod as 'Nome do Produto',**
- ❑ **produtos_quarto.quantidade_pq as 'Quantidade Disponível no Quarto'**
- ❑ **FROM Quarto, Produtos_quarto, Produto**
- ❑ **WHERE**
- ❑ **(quarto.cod_qua = produtos_quarto.cod_qua) AND**
- ❑ **(produtos_quarto.cod_prod = produto.cod_prod) AND (quarto.cod_qua = 2);**

Análise do Código

- ❑ Observe que o **nº 2** na última **condição** da consulta determina de **qual quarto** será listado os produtos.
- ❑ Conforme mudamos o número do quarto (*chave primária*) o resultado da consulta também é modificado;
- ❑ Teste este código no Script BD_Hotel 1.0 disponível no AVA;

Análise do Código - Desempenho

- A execução desse SELECT com 03 tabelas por **01 usuário da camada de aplicação** não afetará o desempenho do sistema, porém, se 500 usuários acessarem a mesma função ao mesmo tempo teremos um grande tráfego de informações pela rede, pois os dados do SELECT irão navegar na rede do Cliente para o Servidor;
- O que não ocorre se esse SELECT for usado dentro de um **procedimento armazenado**. Neste caso o usuário (Aplicação) enviará apenas a **chamada** do procedimento pela rede e todo o código do SELECT ficará armazenado no **servidor** do banco de dados (SGBD);

Análise do Código - Segurança

- Em sistemas com arquitetura **Cliente-Servidor**, nada adianta ter um sistema muito bem programado e um computador potente se a rede por trafega as informações entre o Cliente (Aplicação) e o Servidor (SGBD) estiver congestionada, ou seja, **lenta**;
- Outro conceito importante é que **apenas o Servidor** conhece o código SQL solicitado. Com o procedimento esse código **não irá trafegar** na rede, impossibilitando uma interceptação desde código.
- Um **comando SQL** contem informações **sensíveis** referente a arquitetura do banco de dados que não devem ser divulgadas ou interceptadas;

Sintaxe do Procedure

- ❑ A sintaxe de criação de um **procedimento armazenado** é descrita abaixo:
- ❑ **CREATE PROCEDURE** nome_do_procedimento
 - ❑ ([parâmetros de entrada e/ou saída])
- ❑ **BEGIN**
 - ❑ comandos em SQL
- ❑ **END;**

Análise da Estrutura do Comando

- ❑ **NOME DO PROCEDIMENTO:** Deve-se inserir o nome desejado para o procedimento, usando a mesma regra utilizada nos nomes das tabelas;
- ❑ O nome do procedimento deve estar relacionado a sua **finalidade**;
- ❑ **Exemplos**
 - SelecionarProdutosVendidosMes();
 - VenderProdutos();
 - ListarClientes();

Análise da Estrutura do Comando

- ❑ **PARÂMETROS DE ENTRADA E/OU SAÍDA:** Um parâmetro é um ou vários **atributos** utilizados como **entrada** ou **saída** de dados do Procedimento;
- ❑ Cada atributo utilizado como parâmetro deve possuir um **nome** e um **tipo de dado**;
- ❑ Cada parâmetro é separado por **vírgula** e entre **parêntese**;
- ❑ **Exemplo:** (nome VARCHAR (100), dataNasc DATE, código INT)

Análise da Estrutura do Comando

- ❑ **TIPO DE PARÂMETRO:** Para definir um parâmetro como de **Entrada** ou de **Saída** deve-se utilizar a notação **IN para ENTRADA** ou **OUT para SAÍDA**;
- ❑ **Atenção:** Se nada for informado, o MySQL entende que o parâmetro é de ENTRADA. Assim, se o parâmetro for de ENTRADA, não é obrigatório o uso do IN
- ❑ **Exemplo:** (**código** INT, **nome** VARCHAR (100), **OUT resultado** INT)

Análise da Estrutura do Comando

- **BEGIN e END:** Indica onde inicia e termina os comandos SQL;
- Os comandos DML são os códigos mais comuns utilizados nos procedimentos, mas também podem ser utilizados comandos DDL;
- Nos procedimentos são utilizadas estruturas comparação (IF e ELSE) e estruturas de repetição (WHILE) conhecidas das linguagens de programação;

Chamada de Procedimento

- Para chamar um procedimento deve-se utilizar o código:

CALL nome_do_procedimento (*dados conforme parâmetros*);

- **Exemplos:**

- CALL buscarClientePorNome ('João da Silva');
- CALL buscarClientePorCodigo (50);
- CALL inserirCliente (null, 'João da Silva', 18, '1990-01-01');
- CALL deletarClientePorCodigo (10);

Chamada de Procedimento

- ❑ **ATENÇÃO:** Caso o procedimento chamado possua um **parâmetro de saída**, deve ser criada uma **variável global** para receber o resultado do parâmetro de saída;
- ❑ Para criar uma variável global basta utilizar o símbolo **@** e o nome da variável;
- ❑ A variável global pode ser **consultada** através de um SELECT ou utilizada como um **atributo de entrada** em outro procedimento;
- ❑ **Exemplo:**
 - ❑ CALL calcularQuantProdutos (@resultado);
 - ❑ SELECT @resultado

Delimiter \$\$

- ❑ Para utilizar procedimentos é necessário delimitar a onde é INICIADO e FINALIZADO o procedimento;
- ❑ Para isso é preciso utilizar os comandos:
 - Para o *início*: **DELIMITER \$\$**
 - Para o *final*: **\$\$ DELIMITER ;**
- ❑ Isso é feito para que possamos usar o caractere “;” no meio do procedimento e o MySQL execute o procedimento como um bloco e não por partes;
- ❑ **Atenção:** Deve haver um espaço após o DELIMITER final e o ;

Criando Procedimentos

- ❑ Lembra-se daquela **SELECT com 03 tabelas** mostrado no início do material?
- ❑ Podemos criar um procedimento com aquele código. O **código do quarto** informado pelo usuário (Aplicação) será o **parâmetro de entrada** do procedimento;
- ❑ **Veja o código a seguir:**

Exemplo de Procedimento

- ❑ DELIMITER \$\$
- ❑ **CREATE PROCEDURE** MostrarProdutosQuartos(**código** INT)
 - ❑ **BEGIN**
 - ❑ SELECT
 - ❑ quarto.número_qua as 'Número do Quarto',
 - ❑ produto.descrição_prod as 'Nome do Produto',
 - ❑ produtos_quarto.quantidade_pq as 'Quantidade Disponível no Quarto'
 - ❑ FROM Quarto, Produtos_quarto, Produto
 - ❑ WHERE
 - ❑ (quarto.cod_qua = produtos_quarto.cod_qua) AND
 - ❑ (produtos_quarto.cod_prod = produto.cod_prod) AND (quarto.cod_qua = **código**);
 - ❑ **END** \$\$ DELIMITER ;

Chamada de Procedimento

- A execução do **procedimento armazenado** é feita usando o comando: **CALL MostrarProdutosQuartos(*nº do quarto*);**

Exemplo 1: CALL MostrarProdutosQuartos (1);

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
	Número do Quarto	Nome do Produto	Quantidade Disponível no Quarto
▶	100	Agua Mineral Sem Gás 500ml	2
	100	Chocolate em Barra 200gr	5
	100	Cerveja Skol 350ml	2

Result 9 x

Exemplo 2: CALL MostrarProdutosQuartos (2);

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
	Número do Quarto	Nome do Produto	Quantidade Disponível no Quarto
▶	101	Agua Mineral Sem Gás 500ml	2
	101	Cerveja Skol 350ml	1

Result 10 x

Análise do Procedimento

- Observe que o Procedimento irá executar o código que está dentro dele entre o BEGIN e o END e o **resultado irá depender** da informação **digitada** pelo usuário (Aplicação) na chamada do procedimento;
- Vamos analisar **outros exemplos**, aplicados ao sistema do Hotel Dorme Bem:

Exemplo I

- ❑ DELIMITER \$\$
- ❑ **CREATE PROCEDURE** relatórioHospedagemCliente (**codigo INT**)
- ❑ **BEGIN**
 - ❑ SELECT
 - ❑ cliente.nome_cli,
 - ❑ hospedagem.*
 - ❑ FROM
 - ❑ Hospedagem, Cliente
 - ❑ WHERE
 - ❑ (cliente.cod_cli = hospedagem.cod_cli) AND (cliente.cod_cli = **codigo**);
- ❑ **END \$\$ DELIMITER ;**

Sobre os Procedimentos

- ❑ O **número de parâmetros** de um procedimentos é **ilimitado**. Podemos utilizar vários parâmetros de entrada e/ou de saída no mesmo procedimento;
- ❑ Também podemos utilizar **qualquer tipo de comando SQL** dentro de um parâmetro como por exemplo os comandos DDL (**CREATE, ALTER, DROP**) e os DML (**INSERT, UPDATE, DELETE e SELECT**);
- ❑ Veja os exemplos a seguir:

Exemplo II

- ❑ DELIMITER \$\$
- ❑ **CREATE PROCEDURE** atualizarRendaCliente (**renda** **FLOAT**,
codigo_cliente **INT**)
- ❑ **BEGIN**
 - ❑ UPDATE Cliente SET rendafamiliar_cli = **renda** WHERE cod_cli = **codigo_cliente**;
 - ❑ Select 'Cliente atualizado com sucesso!' as Confirmação;
- ❑ **END**
- ❑ \$\$ DELIMITER ;

- ❑ **CALL** atualizarRendaCliente (2500, 10);

Mensagem de Confirmação

- Observe que o **Exemplo II** foi utilizado uma **mensagem de confirmação** após o comando **UPDATE** através do comando:
 - *Select 'Cliente atualizado com sucesso' as Confirmação;*
- Esse tipo de mensagem **é muito importante** para retornar e informar para o usuário (Aplicação) que o procedimento foi **executado corretamente** e a **ação programada (*UPDATE*)** foi **realizada**;

Exemplo III

- ❑ DELIMITER \$\$
- ❑ **CREATE PROCEDURE** inserirEndereco (**rua** VARCHAR (100), **numero** INT, **bairro** VARCHAR (100), **cidade** VARCHAR (100), **estado** VARCHAR (100))
- ❑ **BEGIN**
 - ❑ INSERT INTO Endereco VALUES (null, rua, numero, bairro, cidade, estado);
 - ❑ Select 'Insert realizado com sucesso!' as Confirmacao;
- ❑ **END**
- ❑ \$\$ DELIMITER ;

- ❑ **CALL** inserirEndereco ('Rua da Paz', 500, 'Centro', 'Ji-Paraná', 'RO');

Sobre Procedimentos

- Também é possível criar procedimentos **SEM PARÂMETROS**;
- Veja o exemplo a seguir:

Exemplo IV

- ❑ DELIMITER \$\$
- ❑ **CREATE PROCEDURE** listarTodosClientes ()
- ❑ **BEGIN**
- ❑ select * from cliente;
- ❑ **END**
- ❑ \$\$ DELIMITER ;

- ❑ **CALL** listarTodosClientes();

Procedimentos com Parâmetros de Saída






- Dependendo da aplicação desenvolvida, pode ser necessário que um procedimento **retorne um valor** que deverá ser usado por **outro procedimento**;
- Neste caso o procedimento deverá ter **um parâmetro de saída**;
- Para que o retorno possa ser **captado** pelo SGBD e usado por outro procedimento ou código SQL é necessário que o mesmo seja **armazenado em uma variável GLOBAL**;
- Veja o exemplo a seguir:

Exemplo V

- ❑ DELIMITER \$\$
- ❑ **CREATE PROCEDURE** MostrarConsumoCliente (**codigo_cli** INT, **OUT total** **DOUBLE**)
- ❑ **BEGIN**
 - ❑ SELECT
 - ❑ SUM(hospedagem.valorConsumo_hosp + hospedagem.valorDiárias_hosp)
 - ❑ INTO **total**
 - ❑ FROM Cliente, Hospedagem
 - ❑ WHERE (Cliente.cod_cli = Hospedagem.cod_cli) AND
 - ❑ (Cliente.cod_cli = **codigo_cli**);
- ❑ **END** \$\$ DELIMITER ;

Chamada de Procedimento com Parâmetros

- CALL **MostrarConsumoCliente** (8, @resultado);
- SELECT @resultado;

Result Grid				Filter Rows: <input type="text"/>	Export: 	Wrap Cell Content: 
	@resultado					
	225					

Result 7 x

Análise do Procedimento

- ❑ Podemos ver que o procedimento foi criado com dois parâmetros, um de **ENTRADA** chamado de **codigo_cli** e outro de **SAÍDA** chamado de **total**;
- ❑ Os parâmetros de um procedimento são chamadas de **variáveis locais** e só existem durante a execução de um procedimento;
- ❑ Perceba que não é necessário colocar a cláusula IN para definir o parâmetro ENTRADA, mas deve-se usar a cláusula OUT na definição do parâmetro de SAÍDA;
- ❑ Observe que no comando SELECT é utilizado a instrução **INTO total**, cuja função é direcionar o **resultado** da função **SUM** para **dentro** da variável **total**;
- ❑ ATENÇÃO: Só pode ser enviado 01 registro para **cada** variável de saída;

Análise do Procedimento

- Na **execução** do procedimento foi consultando o valor total dos produtos consumidos pelo cliente com o código **8** (**informado na chamada do procedimento**) e valor foi armazenado numa **variável local** de saída chamada **total**;
- Na chamada do procedimento foi utilizada uma **variável global** chamada **@resultado** para “pegar” o valor armazenado na variável **total**;
- Observe a utilização do comando SELECT para verificar o conteúdo da variável global **@resultado**;

Exclusão de Procedimento

- ❑ Para excluir um **procedimento armazenado** do banco de dados, você deve utilizar o comando DROP;
- ❑ Sintaxe:
- ❑ **DROP PROCEDURE nome_do_procedimento;**
- ❑ Exemplo:
- ❑ **DROP PROCEDURE relatórioHospedagemCliente;**

Exercícios para Praticar – Utilize o Script BD_Hotel

1. Crie um procedimento para obter a lista de produtos disponíveis no hotel com o estoque maior do que um número recebido no parâmetro de entrada;
2. Crie um procedimento com passagem de parâmetros de entrada para cadastrar um novo Produto;
3. Crie um procedimento que retorne por uma variável de saída o número total de clientes cadastrados no sistema do hotel;
4. Crie um procedimento que atualize para **mais** o valor dos produtos no estoque em **X%** (**X** é o parâmetro de entrada);
5. Crie um procedimento para somar o valor do consumo e o valor da diária e atribuir ao valor total da hospedagem, receba a hospedagem como parâmetro de entrada;