**Introdução aos Stored Procedures**

Quando desenvolvemos aplicações que acessam [**banco de dados**](https://www.devmedia.com.br/cursos/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.

Dependendo da rotina a ser executada, 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.

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. Assim, considerando que as máquinas servidoras geralmente têm configurações de hardware mais robustas, enquanto nada se pode garantir com relação às máquinas clientes, essa pode ser uma “saída” a se considerar.

A questão em que se esbarra é: como executar várias ações no banco de dados a partir de uma única instrução? A resposta que trazemos neste artigo para essa pergunta é: **Stored Procedures** (ou Procedimentos Armazenados, em português).

Stored 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.

A seguir é apresentada uma situação hipotética para que possamos entender melhor o conceito e importância dos procedimentos armazenados.

**Usar ou Não usar Stored Procedures**

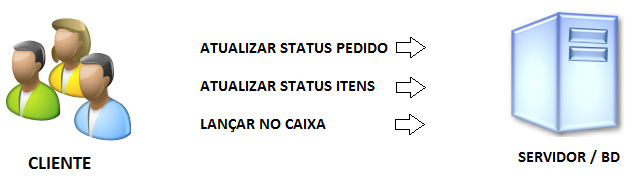
Para exemplificar o funcionamento dos **Stored Procedures**e comparar a execução de uma rotina utilizando e não utilizando essa técnica, tomemos como base o seguinte contexto de uma aplicação comercial.

* O cliente faz um pedido, no qual são inseridos itens;
* O pedido (bem como os itens) permanece com status “PENDENTE” até ser confirmado;
* O operador confirma o pedido, registrando o movimento no livro caixa.

Até o pedido ser confirmado, nenhum lançamento é feito no livro caixa, então é preciso ter uma rotina de confirmação do pedido, que deve executar as seguintes ações:

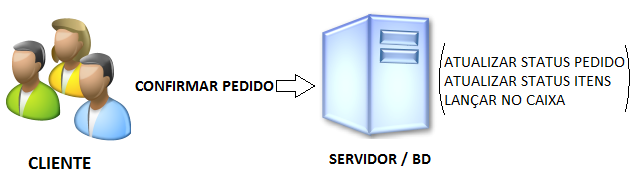
* Atualizar o status do pedido;
* Atualizar o status dos itens do pedido;
* Lançar o valor do pedido no caixa.

Temos então pelo menos três instruções de atualização e/ou inserção. Poderíamos representar essa situação graficamente pela **Figura 1**.



**Figura 1:** Execução da rotina sem stored procedure

Por outro lado, poderíamos agrupar essas três instruções no corpo de um procedimento e chamá-lo a partir da aplicação uma única vez. As ações de update/insert/delete, a partir daí, ficariam por conta do servidor. A representação gráfica desse modelo é mostrada na **Figura 2** (considerando um procedure teoricamente chamado de “CONFIRMAR PEDIDO”).



**Figura 2:** Execução da rotina usando stored procedure

Então, tendo entendido o funcionamento dos procedimentos armazenados, podemos citar as principais vantagens e desvantagens de seu uso: 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 e invocando Stored Procedures no MySQL**

Entrando no real foco deste artigo, será explicado a seguir como trabalhar com procedures no [**banco de dados MySQL**](https://www.devmedia.com.br/curso/curso-completo-de-mysql/281), iniciando pela sintaxe utilizada para criação desse tipo de objeto, que pode ser vista na **Listagem 1**.

**Listagem 1:** Sintaxe para criação de stored procedures no MySQL

DELIMITER $$

CREATE PROCEDURE nome\_procedimento (parâmetros)

BEGIN

/\*CORPO DO PROCEDIMENTO\*/

END $$

DELIMITER ;

Onde consta “nome\_procedimento”, deve-se informar o nome que identificará o procedimento armazenado. Este nome segue as mesmas regras para definição de variáveis, não podendo iniciar com número ou caracteres especiais (exceto o underline “\_”).

Os “parâmetros” são opcionais e, caso não sejam necessários, devem permanecer apenas os parênteses vazios na declaração do procedure. Para que um procedimento receba parâmetros, é necessário seguir certa sintaxe (dentro dos parênteses), apresentada abaixo (**Listagem 2**).

**Listagem 2:** Sintaxe de declaração de parâmetros em stored procedures

(MODO nome TIPO, MODO nome TIPO, MODO nome TIPO)

Aqui, é válido iniciar a explicação pelos itens mais simples.

O “nome” dos parâmetros também segue as mesmas regras de definição de variáveis.

O “TIPO” nada mais é que do tipo de dado do parâmetro (int, varchar, decimal, etc).

O “MODO” indica a forma como o parâmetro será tratado no procedimento, se será apenas um dado de entrada, apenas de saída ou se terá ambas as funções. Os valores possíveis para o modo são:

* IN: indica que o parâmetro é apenas para entrada/recebimento de dados, não podendo ser usado para retorno;
* 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.

Outro ponto que merece destaque na **Listagem 1** é o uso do comando DELIMITER. Por padrão o MySQL utiliza o sinal de ponto e vírgula como delimitador de comandos, separando as instruções a serem executadas. No entanto, dentro do corpo do stored procedure será necessário separar algumas instruções internamente utilizando esse mesmo sinal, por isso é preciso inicialmente alterar o delimitador padrão do MySQL (neste caso, para $$) e ao fim da criação do procedimento, restaurar seu valor padrão.

Tendo criado o procedure, chamá-lo é bastante simples. Para isso fazemos uso da palavra reservada CALL, como mostra o código da **Listagem 3**.

**Listagem 3:** Sintaxe para chamar um stored procedure

CALL nome\_procedimento(parâmetros);

A seguir temos um exemplo de uso de cada tipo de parâmetro.

**Listagem 4:** Usando parâmetro de entrada

DELIMITER $$

CREATE PROCEDURE Selecionar\_Produtos(IN quantidade INT)

BEGIN

SELECT \* FROM PRODUTOS

LIMIT quantidade;

END $$

DELIMITER ;

Esse procedimento tem por função fazer um select na tabela PRODUTOS, limitando a quantidade de registros pela quantidade recebida como parâmetro. Assim, caso desejássemos selecionar dois registros dessa tabela, poderíamos usar o procedure como mostra a **Listagem 5**.

**Listagem 5:** Chamando procedure com parâmetro de entrada

CALL Selecionar\_Produtos(2);

O próximo código mostra um exemplo de recebimento e retorno de parâmetro de saída.

**Listagem 6:** Usando parâmetro de saída

DELIMITER $$

CREATE PROCEDURE Verificar\_Quantidade\_Produtos(OUT quantidade INT)

BEGIN

SELECT COUNT(\*) INTO quantidade FROM PRODUTOS;

END $$

DELIMITER ;

A função desse procedimento é retornar a quantidade de registros da tabela PRODUTOS, passando esse valor para a variável de saída “quantidade”. Para isso foi utilizada a palavra reservada INTO.

Para chamá-lo, usamos um símbolo de arroba (@) seguido do nome da variável que receberá o valor de saída. Feito isso, a variável poderá ser usada posteriormente, como vemos na **Listagem 7**.

**Listagem 7:** Chamando procedure com parâmetro de saída

CALL Verificar\_Quantidade\_Produtos(@total);

SELECT @total;

Ao executar a segunda linha, teremos como retorno o valor da variável @total, que será preenchida no procedure.

O terceiro exemplo mostra um stored procedure chamado Elevar\_Ao\_Quadrado, que recebe uma variável e a altera, definindo-a como o seu próprio valor elevado à segunda potência.

**Listagem 8:** Usando parâmetro de entra e saída

DELIMITER $$

CREATE PROCEDURE Elevar\_Ao\_Quadrado(INOUT numero INT)

BEGIN

SET numero = numero \* numero;

END $$

DELIMITER ;

Nesse caso, a mesma variável é usada como entrada e saída, como vemos na chamada da **Listagem 9**.

**Listagem 9:** Chamando procedure com parâmetro de entrada e saída

SET @valor = 5;

CALL Elevar\_Ao\_Quadrado(@valor);

SELECT @valor;

**Usando variáveis no corpo do procedimento**

É possível declarar variáveis no corpo dos stored procedures, para isso basta utilizar a seguinte sintaxe:

**Listagem 10:** Sintaxe de declaração de variáveis

DECLARE nome\_variável TIPO DEFAULT valor\_padrao;

A palavra reservada DECLARE é obrigatória e é a responsável por indicar que uma variável será declarada com o nome “nome\_variavel” (que segue as mesmas regras de nomeação de variáveis). O TIPO é o tipo de dados da variável (int, decimal, varchar, etc). A palavra reservada DEFAULT é opcional e deve ser usada quando se deseja definir um valor inicial (valor\_padrao) para a variável.

A declaração das variáveis deve ser feita logo no início do corpo do procedure, para aquelas que serão utilizadas em todo o procedimento, ou dentro de um bloco BEGIN-END específico que limite seu escopo.

Para definir um valor para uma variável, usamos as palavras reservadas SET (no caso de passagem direta de valor, como na **Listagem 8**) ou INTO (no caso de associação de valores dentro de consultas, como na **Listagem 6**).

Outro ponto importante de se citar é o ESCOPO das variáveis, que define em que pontos elas são reconhecidas. Uma variável definida dentro de um bloco BEGIN/END é válida somente dentro dele, ou seja, após o END ela já não é mais reconhecida. Assim, é possível definir várias variáveis com o mesmo nome, mas dentro de blocos BEGIN/END distintos.

Por sua vez, variáveis cujo nome inicia com arroba (@), são chamadas variáveis de sessão, e são válidas enquanto durar a sessão (exemplo da Listagem 9).

**Excluindo procedures**

Para excluir uma procedure no MySQL basta utilizar o comando abaixo:

**Listagem 11:** Excluindo procedure

DROP PROCEDURE Listar\_Funcionarios;

Há também a alternativa a seguir:

**Listagem 12:** Excluindo procedure

DROP PROCEDURE IF EXISTS Listar\_Funcionarios;

Ambos os comandos excluem uma stored procedure que tenha sido criada anteriormente.

**Exibindo procedures**

No MySQL temos um comando que exibe todas as stored procedures criadas:

**Listagem 13:** Listando procedures

SHOW PROCEDURE STATUS;

Ao executar o comando acima aparecerá uma lista com todas as stored procedures que foram criadas anteriormente.

**Conclusão**

Nesse artigo vimos como funcionam os Stored Procedures em geral (para qualquer banco) e como trabalhar com essa estrutura no [**banco de dados MySQL**](https://www.devmedia.com.br/curso/mysql-completo/281), usando parâmetros de entrada e saída, declarando variáveis e invocando-os a partir da instrução CALL.

Vale ressaltar que o código do corpo de um procedimento pode conter várias linhas com diversas instruções SQL, aqui apenas foram apresentados exemplos básicos a nível didático.