Procedures

1

Banco de Dados II

Banco de Dados II

Procedures

Introdução aos Stored 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.

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.

Banco de Dados II

Procedures

Introdução aos Stored 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. 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).

Banco de Dados II

Procedures

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.

Banco de Dados II

Procedures

Usar ou Não usar Stored Procedures

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:

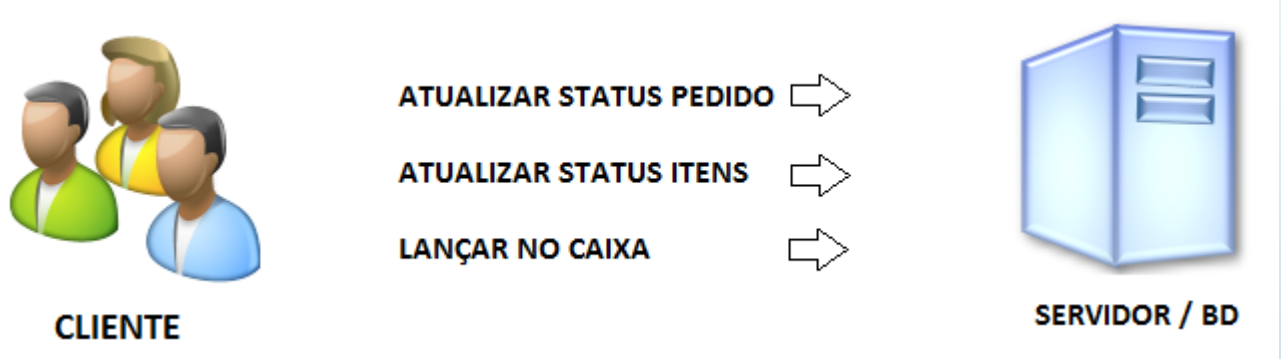
- Atualizar o status do pedido;
- Atualizar o status dos itens do pedido;
- Lançar o valor do pedido no caixa.

Banco de Dados II

Procedures

Usar ou Não usar Stored Procedures

Temos então pelo menos três instruções de atualização e/ou inserção. Poderíamos representar essa situação graficamente pela figura abaixo.



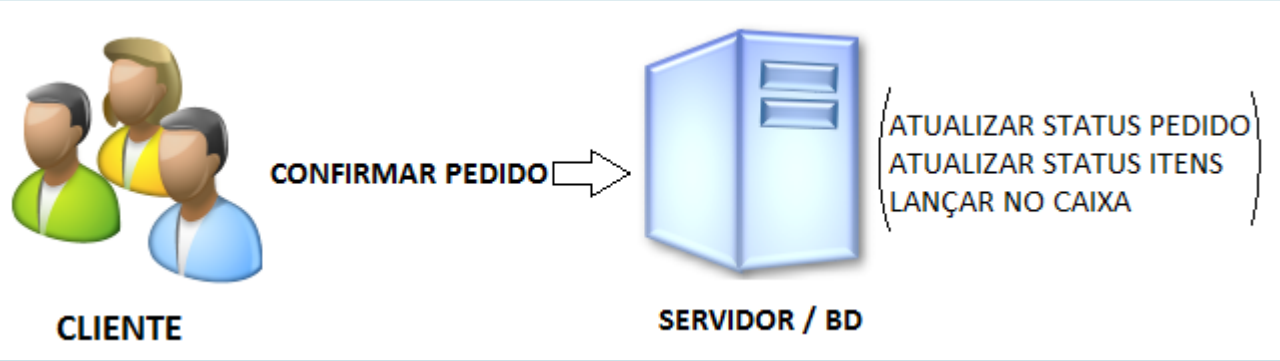
Execução da rotina sem stored procedure

Banco de Dados II

Procedures

Usar ou Não usar Stored Procedures

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 abaixo (considerando um procedure teoricamente chamado de "CONFIRMAR PEDIDO").



Execução da rotina usando stored procedure

Banco de Dados II

Procedures

Usar ou Não usar Stored Procedures

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 SQLpela 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.

Banco de Dados II

Procedures

Usar ou Não usar Stored Procedures

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 SQLpela 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.

Banco de Dados II

Procedures

Criando e invocando Stored Procedures no MySQL

Entrando no real foco deste artigo, será explicado a seguir como trabalhar com ‘ no banco de dados MySQL, iniciando pela sintaxe utilizada para criação desse tipo de objeto, que pode ser vista na listagem abaixo.

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

Sintaxe para criação de stored procedures no MySQL

Banco de Dados II

Procedures
Criando e invocando Stored Procedures no MySQL

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.

```
MODO nome TIPO, MODO nome TIPO, MODO nome TIPO)
```

Sintaxe de declaração de parâmetros em stored procedures

Banco de Dados II

Procedures
Criando e invocando Stored Procedures no MySQL

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.

```
MODO nome TIPO, MODO nome TIPO, MODO nome TIPO)
```

Sintaxe de declaração de parâmetros em stored procedures

Banco de Dados II

Procedures
Criando e invocando Stored Procedures no MySQL

```
MODO nome TIPO, MODO nome TIPO, MODO nome TIPO)
```

Sintaxe de declaração de parâmetros em stored procedures

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).

Sintaxe de declaração de parâmetros em stored procedures

Banco de Dados II

Procedures
Criando e invocando Stored Procedures no MySQL

```
MODO nome TIPO, MODO nome TIPO, MODO nome TIPO)
```

Sintaxe de declaração de parâmetros em stored procedures

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.

Banco de Dados II

Procedures
Criando e invocando Stored Procedures no MySQL

Outro ponto que merece destaque 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 abaixo:

```
CALL nome_procedimento(parâmetros);
```

Sintaxe para chamar um stored procedure

Banco de Dados II

Procedures

Criando e invocando Stored Procedures no MySQL

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

```
DELIMITER $$

CREATE PROCEDURE Selecionar_Produtos(IN quantidade INT)
BEGIN
    SELECT * FROM PRODUTOS
    LIMIT quantidade;
END $$
DELIMITER ;

CALL nome_procedimento(parâmetros);
```

Usando parâmetro de entrada

Banco de Dados II

Procedures
Criando e invocando Stored Procedures no MySQL

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 abaixo:

```
CALL Selecionar_Produtos(2);
```

Chamando procedure com parâmetro de entrada

Banco de Dados II

Procedures
Criando e invocando Stored Procedures no MySQL

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

```
DELIMITER $$

CREATE PROCEDURE Verificar_Quantidade_Produtos(OUT
quantidade INT)
BEGIN
    SELECT COUNT(*) INTO quantidade FROM PRODUTOS;
END $$
DELIMITER ;
```

Usando parâmetro de saída

Banco de Dados II

Procedures
Criando e invocando Stored Procedures no MySQL

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 abaixo:

```
CALL Verificar_Quantidade_Produtos(@total);
SELECT @total;
```

Chamando procedure com parâmetro de saída

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

Banco de Dados II

Procedures

Criando e invocando Stored Procedures no MySQL

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.

```
DELIMITER $$

CREATE PROCEDURE Elevar_Ao_Quadrado(INOUT numero INT)
BEGIN
    SET numero = numero * numero;
END $$
DELIMITER ;
```

Usando parâmetro de entra e saída

Banco de Dados II

Procedures
Criando e invocando Stored Procedures no MySQL

Nesse caso, a mesma variável é usada como entrada e saída, como vemos na chamada da listagem abaixo:

```
SET @valor = 5;
CALL Elevar_Ao_Quadrado(@valor);
SELECT @valor;
```

Chamando procedure com parâmetro de entrada e saída

Banco de Dados II

Procedures
Usando variáveis no corpo do procedimento

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

```
DECLARE nome_variável TIPO DEFAULT valor_padrao;
```

Sintaxe de declaração de variáveis

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.

Banco de Dados II

Procedures

Usando variáveis no corpo do procedimento

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** ou INTO (no caso de associação de valores dentro de consultas)

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.

Banco de Dados II

Procedures

Excluindo procedures

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

```
DROP PROCEDURE Listar_Funcionarios;
```

Excluindo procedure

Há também a alternativa a seguir:

```
DROP PROCEDURE IF EXISTS Listar_Funcionarios;
```

Excluindo procedure

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

Banco de Dados II

Procedures
Exibindo procedures

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

```
SHOW PROCEDURE STATUS;
```

Excluindo procedure

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

Banco de Dados II

Procedures
Exibindo procedures

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

```
SHOW PROCEDURE STATUS;
```

Excluindo procedure

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

Banco de Dados II

Procedures
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, 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.

Referências

- **Dev Media**

  www.devmedia.com.br/stored-procedures-no-mysql/29030