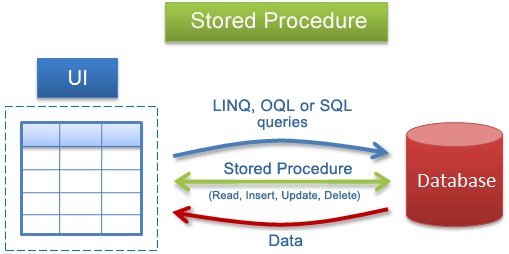
**Stored Procedure (SQL)**

Stored Procedure é um conjunto de comandos, ao qual é atribuído um nome. Este conjunto fica armazenado no Banco de Dados e pode ser chamado a qualquer momento tanto pelo SGBD (sistema Gerenciador de Banco de Dados) quanto por um sistema que faz interface com o mesmo.



A utilização de Stored Procedures é uma técnica eficiente de executarmos operações repetitivas. Ao invés de digitar os comandos cada vez que determinada operação necessite ser executada, criamos um Stored Procedure e o chamamos. Em um Stored Procedure também podemos ter estruturas de controle e decisão, típicas das linguagens de programação. Em termos de desenvolvimento de aplicações, também temos vantagens com a utilização de Stored Procedures.

CREATE PROCEDURE TESTE

AS

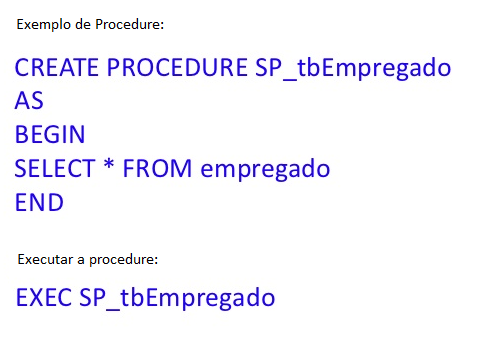
BEGIN

SELECT

‘O FAMOSO HELLO WORLD!’

END

Perceba que os comandos de início e término de bloco, **BEGIN** e **END** respectivamente, são obrigatórios no início e fim do comando.



Bom, pode receber parâmetros, e utilizar eles em instruções SQL que serão executadas dentro da Stored Procedure:

CREATE PROCEDURE TESTE @NOVO INT

AS

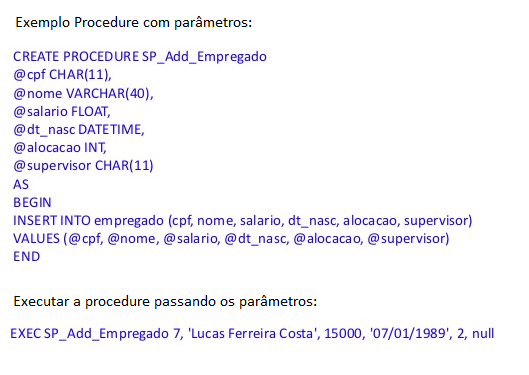
BEGIN

UPDATE TABELA1 SET CAMPO1 = ‘NOVO\_VALOR’

WHERE CAMPO2 = @NOVO

END

Perceba que no exemplo acima , não é utilizado parênteses , pois Stored Procedures é um pouco diferente de funções. Como uma Stored Procedure fica armazenada no banco de dados, ela já é pré-compilada e o SQL Server a executa mais rapidamente.



Outra vantagem das Stored Procedures é que um programa chamador, seja ele uma página ASP ou um programa em VB, Delphi, Java, etc, só precisa chamar o nome da Stored Procedure, que pode conter diversos comandos embutidos dentro dela, evitando assim um tráfego de rede maior, resultando em resposta mais rápida.

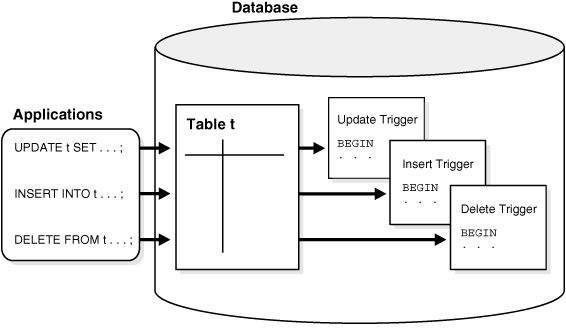
O uso de Stored Procedure é encorajado, mais se deve utilizar este recurso com cuidado, pois se utilizado em excesso o SQL Server pode ser sobrecarregado, mas ao mesmo tempo podemos obter um ganho de performance considerável, dependendo do caso.

**Exercícios:**

1. Escreva uma procedure onde faça a soma de dois números.  
  
Create procedure exercicio01  
@num1 float,  
@num2 float,  
@resposta float output  
as set @resposta = (@num1 + @num2)  
  
  
2. Escreva uma procedure onde entre com quatro valores e calcule a média.  
  
Create procedure exercicio02  
@num1 float,  
@num2 float,  
@num3 float,  
@num4 float,  
@resposta float output  
as set @resposta = (@num1 + @num2 + @num3 + @num4) / 4  
  
  
3. Escreva uma procedure se a soma das rendas dos clientes for menor do que 345,56 aplique um aumento de 5,14% para todos os clientes. Caso contrário 10,23%.  
  
use infonew  
go  
create procedure exercicio03  
as declare @percentual float  
if (select sum (renda\_cliente) from clientes) 567.87  
set @percentual = 0.8737  
else  
set @percentual = 0.9119  
update funcionarios  
SET salario\_funcionario = @percentual \* salario\_funcionario  
  
  
5. Escreva uma procedure se o maior valor unitário do produto for maior ou igual a 1200 aplique uma redução de 9,07% para todos os produtos. Caso contrário uma redução de 4,02%.  
  
Create procedure exercicio05  
as declare @percentual float  
if (select max(valor\_unitarioproduto) from produtos) > 1200  
set @percentual = 0.9093  
else  
set @percentual = 0.9598  
update produtos  
SET valor\_unitarioproduto = @percentual \* valor\_ unitarioproduto  
  
  
6. Escreva uma procedure onde entre com dois parâmetros, onde primeiro indica o código do cliente e o segundo indica o valor da porcentagem de aumento.  
  
CREATE PROCEDURE exercicio06  
( @cod\_cliente int, @percentual float )  
AS  
UPDATE CLIENTES  
SET RENDA\_CLIENTE = RENDA\_CLIENTE \* @percentual  
WHERE COD\_CLIENTE = @cod\_cliente  
  
  
7. Escreva uma procedure onde entre com dois parâmetros, onde primeiro indica o código do produto e o segundo indica o valor da porcentagem de aumento.  
CREATE PROCEDURE exercicio07  
( @cod\_produto INT, @percentual float )  
AS  
UPDATE PRODUTOS  
SET VALOR\_UNITARIOPRODUTO = VALOR\_ UNITARIOPRODUTO \* @percentual  
WHERE COD\_PRODUTO = @cod\_produto

**Triggers (SQL)**

Trigger é um recurso de programação executado sempre que o evento associado ocorrer. Trigger é um tipo especial de procedimento armazenado, que é executado sempre que há uma tentativa de modificar os dados de uma tabela que é protegida por ele. É muito utilizada para ajudar a manter a consistência dos dados ou para propagar alterações em um determinado dado de uma tabela para outras. Um bom exemplo é um gatilho criado para controle de quem alterou a tabela, nesse caso, quando a alteração for efetuada, a trigger é "disparada" e grava em uma tabela de histórico de alteração, o usuário e data/hora da alteração.



**Prós e Contras das Triggers**

**Os principais pontos positivos sobre os triggers são:**

* Parte do processamento que seria executado na aplicação passa para o banco, poupando recursos da máquina cliente.
* Facilita a manutenção, sem que seja necessário alterar o código fonte da aplicação.

**Já contra sua utilização existem as seguintes considerações:**

* Alguém que tenha acesso não autorizado ao banco de dados poderá visualizar e alterar o processamento realizado pelos gatilhos.
* Requer maior conhecimento de manipulação do banco de dados (SQL) para realizar as operações internamente.

A sintaxe dos comandos para criar um novo trigger no MySQL é a seguinte:

CREATE TRIGGER nome momento evento

ON tabela

FOR EACH ROW

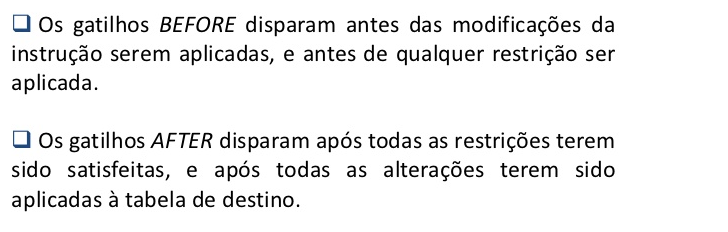
BEGIN

/\*corpo do código\*/

END

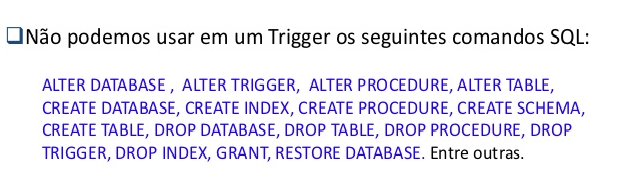
Onde se tem os seguintes parâmetros:

* nome: nome do trigger, segue as mesmas regras de nomeação dos demais objetos do banco.
* momento: quando o trigger será executado. Os valores válidos são BEFORE (antes) e AFTER (depois).



* evento: evento que vai disparar o trigger. Os valores possíveis são INSERT, UPDATE e DELETE. Vale salientar que os comandos LOAD DATA e REPLACE também disparam os eventos de inserção e exclusão de registros, com isso, os gatilhos também são executados.
* tabela: nome da tabela a qual o gatilho está associado.

Não é possível criar mais de um trigger para o mesmo evento e momento de execução na mesma tabela. Por exemplo, não se pode criar dois triggers AFTER INSERT na mesma tabela.



**Os registros NEW e OLD**

Como os triggers, são executados em conjunto com operações de inclusão e exclusão, é necessário poder acessar os registros que estão sendo incluídos ou removidos. Isso pode ser feito através das palavras NEW e OLD. Em triggers executados após a inserção de registros, a palavra reservada NEW dá acesso ao novo registro. Pode-se acessar as colunas da tabela como atributo do registro NEW, como veremos nos exemplos.

O operador OLD funciona de forma semelhante, porém em triggers que são executados com a exclusão de dados, o OLD dá acesso ao registro que está sendo removido.

**Utilização do trigger**

Para exemplificar e tornar mais clara a utilização de triggers, é simulado a seguinte situação: um mercado que, ao realizar vendas, precisa que o estoque dos produtos seja automaticamente reduzido. A devolução do estoque deve também ser automática no caso de remoção de produtos da venda.

Como se trata de um ambiente hipotético, teremos apenas duas tabelas de estrutura simples, cujo script de criação é mostrado na listagem a seguir.

CREATE TABLE produtos

( referencia VARCHAR(3) PRIMARY KEY, descricao VARCHAR(50) UNIQUE, estoque INT NOT NULL );

INSERT INTO Produtos VALUES ('1', 'Lasanha', 10);

INSERT INTO Produtos VALUES ('2', 'Morango', 5);

INSERT INTO Produtos VALUES ('3', 'Farinha', 15);

CREATE TABLE itensvenda

( venda INT, produto VARCHAR(3), quantidade INT );

Ao inserir e remover registro da tabela itensvenda, o estoque do produto referenciado deve ser alterado na tabela produtos. Para isso, serão criados dois triggers: um AFTER INSERT para dar baixa no estoque e um AFTER DELETE para fazer a devolução da quantidade do produto.

CREATE TRIGGER tgr\_itensvenda\_insert AFTER INSERT

ON itensvenda

FOR EACH ROW

BEGIN

UPDATE produtos SET estoque = Estoque - NEW.quantidade

WHERE referencia = NEW.produto;

END;

CREATE TRIGGER tgr\_itensvenda\_delete AFTER DELETE

ON itensvenda

FOR EACH ROW

BEGIN

UPDATE produtos SET estoque = estoque + OLD.quantidade

WHERE referencia = OLD.produto;

END;

No primeiro trigger, foi utilizado o registro NEW para obter as informações da linha que está sendo inserida na tabela. O mesmo é feito no segundo trigger, onde se obtém os dados que estão sendo apagados da tabela através do registro OLD. Tendo criado os triggers, pode-se testá-los inserindo dados na tabela itensvenda. Nesse caso, é simulado uma venda de número 1 que contem três unidades do produto 1, uma unidade do produto 2 e cinco unidades do produto 3.

Insere dados na tabela.

INSERT INTO itensvenda VALUES (1, '1',3);

INSERT INTO itensvenda VALUES (1, '2',1);

INSERT INTO itensvenda VALUES (1, '3',5);

Nota-se que o estoque dos produtos foi corretamente reduzido, de acordo com as quantidades "vendidas". Agora para testar o trigger da exclusão, será removido o produto 1 dos itens vendidos. Com isso, o seu estoque deve ser alterado para o valor inicial, ou seja, 10.

DELETE FROM itensvenda WHERE venda = 1 AND produto = '1';

Em ambientes reais, triggers podem ser utilizados para operações mais complexas, por exemplo, antes de vender um item, verificar se há estoque disponível e só então permitir a saída do produto.

Outros exemplos:

