PROCEDIMENTOS ARMAZENADOS
(Stored Procedures)

1. Introdução

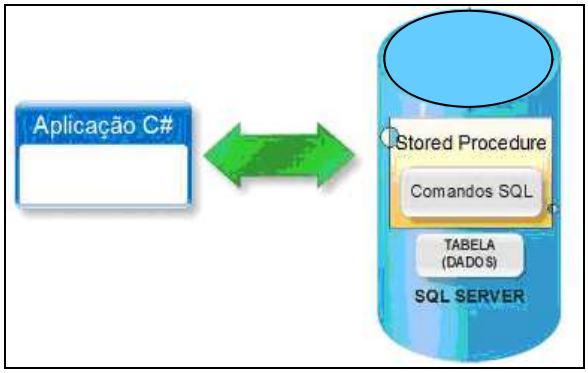
Stored Procedure é um conjunto de comandos, ao qual é atribuído um nome. Este conjunto fica armazenado no Banco de Dados e pode ser chamado a qualquer momento tanto pelo SGBD (sistema Gerenciador de Banco de Dados) quanto por um sistema que faz interface com o mesmo.

Em uma *Stored Procedure*, além dos comandos SQL comuns, também podemos ter estruturas de decisão e repetição, típicas das linguagens de programação.

2. Algumas razões para usar Stored Procedures?

- SP´s são utilizadas para mover grande parte do código de manipulação de dados para o servidor, isso elimina a transferência de dados do servidor para o cliente, pela rede, para manipulação → redução do tráfego na rede → melhor performance geral.

Acesso ao BD sem us de Stored Procedures



Acesso ao BD com uso de Stored Procedures

- Sempre que uma aplicação cliente envia um comando SQL para o servidor, o comando tem que ter a sintaxe analisada e, a seguir, é submetido a um programa *otimizador* do SGBD para formulação de um plano de execução.
  Já as SP´s são analisadas e otimizadas uma única vez (quando são criadas) → são executadas mais rapidamente que os comandos SQL enviados pelas aplicações.

- SP´s podem economizar tempo de desenvolvimento e manutenção de sistemas pois as sp ´s podem ser compartilhadas por todas as aplicações existentes. A manutenção é mais fácil porque é possível alterar uma sp sem ter que alterar e\ou recompilar cada aplicação cliente.

- Ganho de velocidade de processamento pois geralmente o servidor é uma das máquinas mais poderosas na rede.

- SP´s são muito úteis para efetuar tarefas de processamento periódico e complexo, como processamentos de fechamento de mês.

3. Praticando

3.1. Exemplo 1: Criar e executar uma *procedure*

Execute no SQL SERVER todos os passos abaixo:

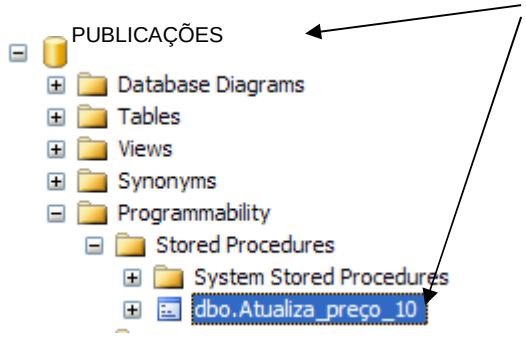
```
comando USE PUBLICAÇÕES (na janela de query para indicar o BD)
ou SELECIONAR O BANCO PUBLICAÇÕES antes de abrir a janela de query
```

Na janela de **query** digite:

```
CREATE PROCEDURE SP_Atualiza_preço_10
AS
BEGIN
UPDATE LIVRO SET PREÇO=PREÇO*1.10
END
```

Clicar no botão **EXECUTE** para **gravar** a *procedure*

Localize a *procedure* criada e armazenada no BD Publicações:



Executar a procedure: (na janela de **query**)

```
EXEC Atualiza_preço_10
```

3.2. Exercício 1: Criar e Executar a seguinte *procedure* (nome SP_Lista_Atualiza):

A) Listar todos os livros cadastrados; atualizar a editora dos livros cujo assunto é banco de dados para a editora Maria Jose Editora; listar novamente todos os livros cadastrados.

B) Alterar a *procedure* A: o segundo *select* deve apresentar apenas título e código editora.

Obs.: quando precisar alterar uma *procedure*, abra uma janela de *query* e digite:
ALTER PROCEDURE NOME_DA_PROCEDURE AS BEGIN *comandos* END
(comandos = todos os comandos da *procedure*)

3.3. Exemplo 2: Criar e executar uma procedure com PARAMETRO de ENTRADA

Execute no SQL SERVER todos passos abaixo:

```
USE LIVROS   /   OU SELECIONAR O BANCO LIVROS ANTES
```


```
CREATE PROCEDURE SP_Atualiza_preço @taxa real
AS
UPDATE LIVRO SET PREÇO=PREÇO+(PREÇO*@taxa/100)
END
```

```
EXEC SP_Atualiza_preço
```

Mensagem de ERRO: **Msg 201, Level 16, State 4, Procedure Atualiza_preço, Line 0. O procedimento ou a função 'Atualiza_preço' espera o parâmetro '@taxa', que não foi fornecido.**

Por que ocorreu a mensagem de erro acima?

```
EXEC SP_Atualiza_preço 5
```

Mensagem: **(5 row(s) affected)**

3.4. Exemplo 3: Excluir uma procedure

Execute:

```
DROP PROCEDURE Atualiza_preço_10
```

3.5. Exercício 2: MULTIPLOS PARÂMETROS

3.5.1. Criar uma procedure para aumentar os preços dos livros conforme a taxa e o assunto fornecidos através de parâmetros.

Obs.: - uma vírgula deve separar cada definição de parâmetro da outra.
- No comando de execução (EXEC) o parâmetro assunto (CHAR) pode ser fornecido com ou sem aspas.

Execute a procedure com EXEC.

3.5.2. Criar uma procedure para excluir livros conforme o assunto (sigla) e a editora (código) fornecidos.

Execute a procedure com EXEC.

3.6. Exemplo 4: Conforme foi dito, as SP também possuem estruturas típicas das linguagens de programação, tais como IF e **WHILE.**

Obs.: se houver mais de um comando dentro do if/else ou do while, tais comandos deverão estar envolvidos por begin\end).

Exemplo utilizando IF:

```
create procedure SP_UpdatePreço_DeleteLivro @opcao char(1)
as
begin
if @opcao='U'
     update livro set preço=preço-(preço*10/100)
if @opcao='D'
     delete from livro where preço<40
end
```

Execute a procedure com EXEC.

GATILHOS
(Trigger)

1. Introdução

Um gatilho (*trigger*) é um tipo especial de *stored procedure*. Um gatilho (*trigger*) também executa um conjunto de comandos. A principal diferença em relação às *procedures comuns* está no fato dos gatilhos serem **automaticamente executados** sempre que um determinado **evento** (**inclusão, alteração e exclusão**) ocorrer no BD.

Gatilhos (trigger´s) somente são executados automaticamente, ou seja, não podemos executá-las através do comando EXEC.

Uma trigger é sempre associada a uma tabela, entretanto, os comandos que formam a trigger podem acessar dados de outras tabelas do mesmo banco.

Principal aplicação: INCLUIR REGRAS DE NEGÓCIO NO BANCO DE DADOS.

2. Praticando

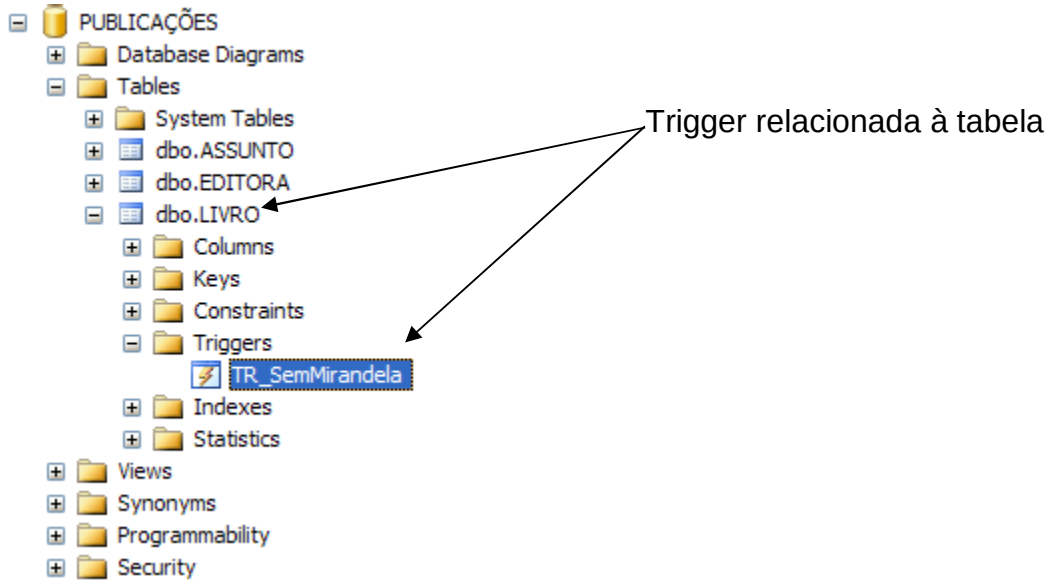
Vamos entender os detalhes na prática.
Execute no SQL SERVER todos os passos abaixo.

2.1. **Exemplo 1:** Suponha que não desejamos ter novos cadastramentos de livros da editora Mirandela (**código 1**).

Criando a trigger (Atenção! Será criada, mas <u>NÃO</u> faz o que se deseja)

```
CREATE TRIGGER TR_SemMirandela ON livro
FOR INSERT
AS
IF EXISTS(SELECT * FROM INSERTED WHERE EDITORA=1)
PRINT 'Atenção! PROIBIDO INCLUIR LIVRO DE MIRANDELA EDITORA'
```

Localizando a trigger:



Testando a trigger:

```
INSERT INTO LIVRO(CODIGO,TÍTULO,PREÇO,ASSUNTO,EDITORA)
VALUES(100,'Java em 5 minutos',10,'P',1)
```

Resultado que obtido:

```
PROIBIDO INCLUIR LIVRO DE MIRANDELA EDITORA
(1 row(s) affected)
```

Qual será o estado da tabela livro agora? O livro da Mirandela foi incluído?

```
SELECT * FROM LIVRO
```

2.2. **Exemplo 2:** No exemplo 1, a simples mensagem de proibição não garante que novos livros da Mirandela Editora não serão cadastrados. Desejamos **impedir** a inclusão de novos livros desta editora (**código 1**).

Alterando a trigger:

```
ALTER TRIGGER TR_SemMirandela ON LIVRO
FOR INSERT
AS
IF EXISTS(SELECT * FROM INSERTED WHERE EDITORA=1)
BEGIN
PRINT 'INSERCAO CANCELADA. NÃO PERMITIDO LIVRO DE MIRANDELA'
ROLLBACK
END
ELSE
PRINT 'INCLUSAO CONFIRMADA'
```

Testando a trigger:

```
INSERT INTO LIVRO(CODIGO,TÍTULO,PREÇO,ASSUNTO,EDITORA)
VALUES(101,'C# em 5 minutos',10,'P',1)
```

Resultado que obtido:

```
INSERCAO CANCELADA. NÃO PERMITIDO LIVRO DE MIRANDELA
Msg 3609, Level 16, State 1, Line 1
A transação foi encerrada no disparador. O lote foi anulado.
```

Qual será o estado da tabela livro agora? O livro da Mirandela foi incluído?

```
SELECT * FROM LIVRO
```

2.3. **Exemplo 3:** Vamos supor que, por algum motivo gerencial\administrativo, a direção da livraria deseja, no momento, IMPEDIR que sejam feitas exclusões na tabela livro.

Devemos criar uma trigger para realizar este controle.

Criando a trigger:

```
CREATE TRIGGER TR_NãoExcluiLivro on LIVRO
FOR DELETE
AS
IF EXISTS(SELECT * FROM DELETED)
BEGIN
 PRINT 'A TABELA LIVRO ESTÁ BLOQUEADA PARA EXCLUSÃO'
 ROLLBACK
END
```

Testando a trigger:

```
DELETE FROM LIVRO WHERE CODIGO=3
```

Resultado que obtido:

```
A TABELA LIVRO ESTÁ BLOQUEADA PARA EXCLUSÃO
Msg 3609, Level 16, State 1, Line 1
A transação foi encerrada no disparador. O lote foi anulado.
```

Qual será o estado da tabela livro agora? O livro de código 3 foi excluído?

```
SELECT * FROM LIVRO
```

Agora tente excluir o livro de código 3 graficamente (abra a tabela, selecione a linha do livro 3 e tecle <delete> e confirme. Qual o resultado?

2.4. **Exercício:** Descreva a função da trigger definida abaixo.

```
CREATE TRIGGER TR_Exercicio on EDITORA
FOR INSERT
AS
DECLARE @NomeEditora varchar(30)
IF EXISTS(SELECT * FROM INSERTED)
BEGIN
 SET @NomeEditora=(SELECT NOME FROM INSERTED)
 PRINT 'TB EDITORA BLOQUEADA PARA INCLUSÃO.TENTATIVA DE INCLUSAO: '
+ @NomeEditora
 ROLLBACK
END
```

Resposta:

Que mensagem o comando abaixo apresentará?

```
INSERT INTO EDITORA(CODIGO,NOME) VALUES(6,'SARAIVA')
```

Resposta:

2.5. A Trigger abaixo impede alterações de linhas da tabela Livro que tenham na coluna editora o código 1 (Mirandela).

```
CREATE TRIGGER TR_SemAlteraçãoMirandela ON livro
FOR UPDATE
AS
IF EXISTS(SELECT editora FROM DELETED WHERE editora=1)
BEGIN
  PRINT 'Não permitido alteração da editora 1 (Mirandela)'
  ROLLBACK
END
```