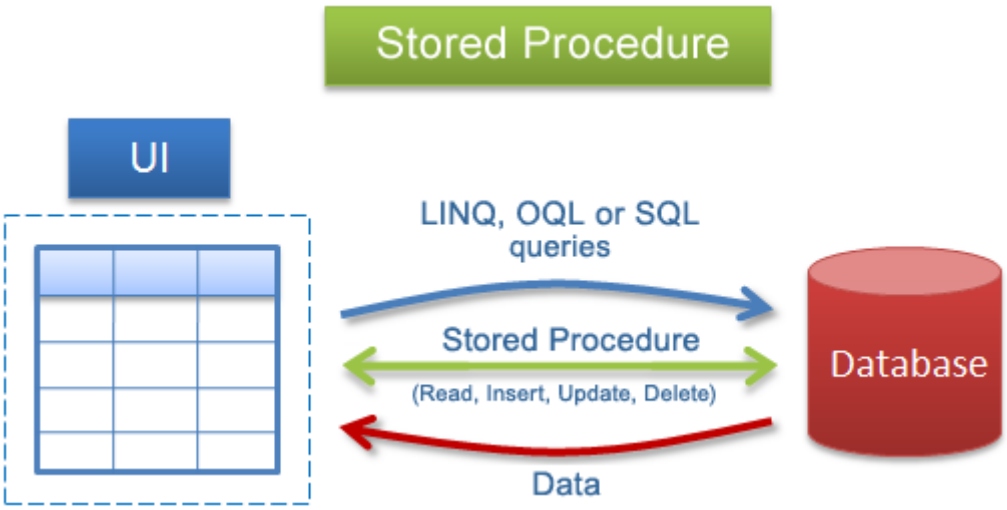
AULA DE FIXAÇÃO SOBRE PROCEDIMENTOS, FUNÇÕES E GATILHOS - 23/05

STORED PROCEDURE

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 execução de 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-se receber parâmetros, e utilizá-los em instruções SQL que serão executadas dentro da Stored Procedure:

```
CREATE PROCEDURE TESTE @PAR1 INT
AS
BEGIN
  UPDATE TABELA1
     SET CAMPO1 = NOVO_VALOR
        WHERE CAMPO2 = @PAR1
END
```

Perceba que no exemplo acima, não é utilizado parênteses, pois Stored Procedure 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 Transact-SQL embutidos dentro dela, evitando assim um tráfego de rede maior, resultando em resposta mais rápida.
Uma Stored Procedure pode ainda retornar valores para a aplicação. Aqui temos um detalhe: o SQL Server permite o retorno de dados em forma de uma tabela após a execução ou um valor de retorno normal. Exemplo:

```
CREATE PROCEDURE TESTE @PAR1 INT
AS
BEGIN
  SELECT @PAR1*@PAR1 AS QUADRADO
END
```

No exemplo acima a aplicação chamadora (cliente) pode capturar o retorno da Stored Procedure através do campo chamado QUADRADO, que contém somente um valor de retorno: o parâmetro elevado ao quadrado.

O uso de Stored Procedure é encorajado, mais deve-se 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.

TRIGGER

Existem várias maneiras de tratarmos ou nos dirigirmos a esse tipo de implementação. Podemos chamar de TRIGGER, gatilhos ou disparadores. No artigo de hoje, usarei o termo mais conhecido entre administradores de Banco de Dados, TRIGGER.

O que são TRIGGERS;

Um 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.

Associados a uma tabela

Os TRIGGERS são definidos em uma tabela específica, que é denominada tabela de TRIGGERS;

Chamados Automaticamente

Quando há uma tentativa de inserir, atualizar ou excluir os dados em uma tabela, e um TRIGGER tiver sido definido na tabela para essa ação específica, ele será executado automaticamente, não podendo nunca ser ignorado.

É parte de uma transação

O TRIGGER e a instrução que o aciona são tratados como uma única transação, que poderá ser revertida em qualquer ponto do procedimento, caso você queria usar "*ROLLBACK"*, conceitos que veremos mais à frente.

Usos e aplicabilidade dos TRIGGERS

- *Impor uma integridade de dados mais complexa do que uma restrição CHECK;*
- *Definir mensagens de erro personalizadas;*
- *Manter dados desnormalizados;*
- *Comparar a consistência dos dados – posterior e anterior – de uma instrução UPDATE;*

Os TRIGGERS são usados com enorme eficiência para impor e manter integridade referencial de baixo nível, e não para retornar resultados de consultas. A principal vantagem é que eles podem conter uma lógica de processamento complexa.

Você pode usar TRIGGERS para atualizações e exclusões em cascata através de tabelas relacionadas em um banco de dados, impor integridades mais complexas do que uma restrição *CHECK,* definir mensagens de erro personalizadas, manter dados desnormalizados e fazer comparações dos momentos anteriores e posteriores a uma transação.

Quando queremos efetuar transações em cascata, por exemplo, um TRIGGER de exclusão na tabela *CLIENTE* do banco de dados *Northwind* pode excluir os registros correspondentes em outras tabelas que possuem registros com os mesmos valores de *ID_CLIENTE* excluídos para que não haja quebra na integridade, como a dito popular “pai pode não ter filhos, mas filhos sem um pai é raro”.

Você pode utilizar os TRIGGERS para impor integridade referencial da seguinte maneira:

- Executando uma ação ou atualizações e exclusões em cascata:

A integridade referencial pode ser definida através do uso das restrições *FOREIGN KEY* e *REFERENCE*, com a instrução *CREATE TABLE.* Os TRIGGERS fazem bem o trabalho de checagem de violações e garantem que haja coerência de acordo com a sua regra de negócios. Se você exclui um cliente, de certo, você terá que excluir também todo o seu histórico de movimentações. Não seria boa coisa se somente uma parte desta transação acontecesse.

Quando mais de um registro for atualizado, inserido ou excluído, você deve implementar um TRIGGER para manipular vários registros.

Criando Triggers

As TRIGGERS são criadas utilizando a instrução CREATE TRIGGER que especifica a tabela onde ela atuará, para que tipo de ação ele irá disparar suas ações seguido pela instrução de conferência para disparo da ação.

E meio a esses comandos, temos algumas restrições para o bom funcionamento. O SQL Server não permite que as instruções a seguir, sejam utilizadas na definição de uma TRIGGER:

- *ALTER DATABASE*
- *CREATE DATABASE*
- *DISKINIT*
- *DISKRESIZE*
- *DROP DATABASE*
- *LOAD DATABASE*
- *LOAD LOG*
- *RECONFIGURE*
- *REATORE DATABASE*
- *RESTORELOG.*

Como funcionam Triggers

- *Como funciona um TRIGGER INSERT*

Quando incluímos, excluímos ao alteramos algum registro em um banco de dados, são criadas tabelas temporárias que passam a conter os registros excluídos, inseridos e também o antes e depois de uma atualização.

Quando você exclui um determinado registro de uma tabela, na verdade você estará apagando a referência desse registro, que ficará, após o *DELETE*, numa tabela temporária de nome *DELETED*. Um TRIGGER implementado com uma instrução *SELECT* poderá lhe trazer todos ou um número de registro que foram excluídos.

Assim como acontece com *DELETE*, também ocorrerá com inserções em tabelas, podendo obter os dados ou o número de linhas afetadas buscando na tabela INSERTED.

Já no UPDATE ou atualização de registros em uma tabela, temos uma variação e uma concatenação para verificar o antes e o depois.

De fato, quando executamos uma instrução *UPDATE*, a "*engine*" de qualquer banco de dados tem um trabalho semelhante, primeiro exclui os dados tupla e posteriormente faz a inserção do novo registro que ocupará aquela posição na tabela, ou seja, um DELETE seguido por um INSERT. Quando então, há uma atualização, podemos buscar o antes e o depois, pois o antes estará na tabela *DELETED* e o depois estará na tabela *INSERTED*.

Nada nos impede de retornar dados das tabelas temporárias (*INSERTED*, *DELETED*) de volta às tabelas de nosso banco de dados, mas atente-se, os dados manipulados são temporários assim como as tabelas e só estarão disponíveis nesta conexão. Após fechar, as tabelas e os dados não serão mais acessíveis.

Trigger INSERT

De acordo com a sua vontade, um TRIGGER por lhe enviar mensagens de erro ou sucesso, de acordo com as transações. Estas mensagens podem ser definidas em meio a um TRIGGER utilizando condicionais *PRINT* sua mensagem personalizada. Por exemplo, vamos criar uma tabela chamada <b mso-bidi-font-weight:"" "="" style="-webkit-tap-highlight-color: transparent;">tbl_usuario, com a qual simularemos um cadastro de usuários que você também poderá criar para fazer os seus testes. A cada inserção de novo usuário, podemos exibir uma mensagem personalizada de sucesso na inserção.

```
1

  DELIMITER // CREATE TRIGGER msgSucess ON tbl_usuario FOR INSERT
  AS IF (SELECT COUNT (*) FROM INSERTED) = 1 PRINT 'O foi inserido
  com sucesso' GO // DELIMITER;
```

Assim que começarmos a pular nossa tabela Existem várias outras abordagens para o uso de TRIGGERS com INSERTS, por exemplo, quando se faz um controle de entrada e saídas de produtos, podemos ter um TRIGGER executando a baixa dos produtos no estoque (entrada) diante daqueles que foram solicitados num pedido (saída). O TRIGGER pode automatizar essa rotina.

Utilizando da mesma lógica pode ser feito para DELETE e UPDATE. Outro exemplo de Trigger:

```
1

  DROP TRIGGER IF EXISTS `grava_log_atualizacao`; DELIMITER //
  CREATE TRIGGER `grava_log_atualizacao` AFTER UPDATE ON `classes`
  FOR EACH ROW INSERT INTO log_update_classe(data, descricao,
  oquefez,quemfez) VALUES(NOW(),'Atualizacao da classe',
  CONCAT('Atualizou a classe: ',old.tipoClasse),@login) //
  DELIMITER ;
```

Apagando uma Trigger

Caso você queira apagar um TRIGGER do banco de dados, utilize DROP TRIGGER mais o nome do TRIGGER que deseja apagar.

```
1

    DROP TRIGGER nomeDaTrigger;
```

DICAS

- *Utilize TRIGGER somente quando necessário.*
- *Quando for utilizar, que ele seja o mais simples possível.*
- *TRIGGERS não tem um bom suporte a muitas transações.*
- *Dependendo de como forem definidas, podem originar problemas com performances.*
- *Minimize o quanto puder instruções ROLLBACK em TRIGGERS.*

Dúvidas críticas ou sugestões comentem abaixo, e bons estudos!!!