Microsoft®
SQL Server®

Alex Sander Resende de Deus

A 25 anos ensinando programação a jovens e adultos.

Apaixonado por tecnologia é atualmente coordenador de cursos na ETEC Albert Einstein. Na FIAP atua como professor na FIAP School, lecionando C#, SQLServer e Desenvolvimento Mobile

Aula 06

O que são Stored Procedures?

Procedimento armazenado ou Stored Procedure é uma coleção de comandos em SQL, que podem ser executadas em um Banco de dados de uma só vez, como em uma função. Os procedimentos armazenados encapsulam tarefas repetitivas, aceitam parâmetros de entrada, são capazes de utilizar os comandos como IF e ELSE, WHILE, LOOP, REPEAT e CASE, além de poderem chamar outros procedimentos armazenados e retornam um valor de status (para indicar aceitação ou falha na execução).

Existem diversos usos para procedimentos armazenados, pois, dentro do procedimento podemos utilizar diversos tipos de comandos como INSERT, UPGRADE, DELETE, MERGE, DROP, CREATE e ALTER assim fornecendo um grande leque de utilidades para procedimentos armazenados.

Um procedimento armazenado também podem ser utilizado para validação de dados e controle de acesso.

Os procedimentos são como funções que serão guardadas no servidor, que podem ou não ser executadas através de um comando “EXEC [nome da procedure] ” (em seu caso sem a necessidade de parâmetros de entrada).

Por ser executada dentro do servidor, o trafego de dados existente na rede é drasticamente reduzido, pois, as únicas coisas que serão passadas pela rede são os valores dos parâmetros de entrada e o nome do procedimento assim otimizando o tempo de execução, diminuindo o uso da CPU e diminuindo a necessidade de memória. Além criar mecanismos de segurança entre a manipulação dos dados do Banco de Dados.

VANTAGENS

Melhoria na performance, já que terá uma menor quantidade de trânsito entre redes;

Pode ser compartilhado entre as aplicações;

Portabilidade;

DESVANTAGENS

Se utilizarmos uma stored procedured podemos ficar bastante dependente da base de dados, com isso, se precisássemos mudar de base por algum motivo, iriamos ter que reescrever todas as storeds procedureds, o que ocasionaria em um grande custo de tempo. Existe ferramentas que consegue fazer uma migração como essa, entretanto nem sempre é funcional.

Praticando no SQLServer

```
create database aulaStoredProcedure

use aulaStoredProcedure


create table alunos(
id int primary key identity (1,1),
nome varchar(50) not null,
email varchar(50),
endereco varchar(100) not null
)
```

```
/*Criação da Stored Procedure*/
create procedure inserirAluno
@nome varchar(50),
@email varchar(50),
@endereco varchar(100)
as
insert into alunos(nome, email, endereco)
values (@nome,@email,@endereco)
select id,nome,email,endereco from alunos
order by id desc

/*Executar a Stored*/
exec inserirAluno 'Maria','maria@fiap.com.br','Rua teste'
```

Momento Hands On

Usando o script do exercício “Game on line”, com personagens e itens, transforme os comandos abaixo em STORED PROCEDURE

a) Cadastre personagens e os itens pertencentes a estes personagens.
b) Selecione todos os personagens e seus itens
c) Selecione um personagem específico e seus itens
d) Selecione um item e todos os personagens que o possua.

Vamos falar sobre TRIGGERS?

Muitas vezes, quando desenvolvemos aplicações com acesso a bancos de dados, utilizamos estes apenas como depósito de informações, sem explorar completamente os recursos que eles nos oferecem. Alguns SGBDs (Sistemas Gerenciadores de Bancos de Dados) dispõem de diversas funcionalidades que, se utilizadas corretamente, podem trazer diversos benefícios, tais como:

Maior facilidade na manutenção do sistema depois de implantando em ambiente de produção;

Ganho de desempenho, quando o banco de dados encontra-se em um servidor com boa capacidade de hardware;

Possibilidade de maior atuação de um DBA (Administrador de Banco de Dados) no desenvolvimento e manutenção do sistema;

Os triggers são um ótimo exemplo desse tipo de funcionalidade que nem sempre é aproveitado pelos desenvolvedores, muitas vezes por não conhecerem ou entenderem o funcionamento dessas estruturas ou por dificuldade com a programação no banco de dados

O termo **trigger** (gatilho em inglês) define uma estrutura do banco de dados que funciona, como o nome sugere, como uma função que é disparada mediante alguma ação.

Geralmente essas ações que disparam os triggers são alterações nas tabelas por meio de operações de inserção, exclusão e atualização de dados (insert, delete e update). Um gatilho está intimamente relacionado a uma tabela, sempre que uma dessas ações é efetuada sobre essa tabela, é possível dispará-lo para executar alguma tarefa.

SINTAXE

```
1  CREATE TRIGGER [NOME DO TRIGGER]
2  ON [NOME DA TABELA]
3  [FOR/AFTER/INSTEAD OF] [INSERT/UPDATE/DELETE]
4  AS
5      --CORPO DO TRIGGER
```

NOME DO TRIGGER: nome que identificará o gatilho como objeto do banco de dados. Deve seguir as regras básicas de Nomenclatura de objetos.

NOME DA TABELA: tabela à qual o gatilho estará ligado, para ser disparado mediante ações de insert, update ou delete.

FOR/AFTER/INSTEAD OF: uma dessas opções deve ser escolhida para definir o momento em que o trigger será disparado. FOR é o valor padrão e faz com o que o gatilho seja disparado junto da ação. AFTER faz com que o disparo se dê somente após a ação que o gerou ser concluída.

INSTEAD OF faz com que o trigger seja executado no lugar da ação que o gerou.

INSERT/UPDATE/DELETE: uma ou várias dessas opções (separadas por vírgula) devem ser indicadas para informar ao banco qual é a ação que disparará o gatilho. Por exemplo, se o trigger deve ser disparado após toda inserção, deve-se utilizar AFTER INSERT.

Praticando no SQLServer

Para exemplificar o uso de gatilhos, tomaremos como cenário uma certa aplicação financeira que contém um controle de caixa e efetua vendas. Sempre que forem registradas ou excluídas vendas, essas operações devem ser automaticamente refletidas na tabela de caixa, aumentando ou reduzindo o saldo.

Vamos então criar as tabelas que utilizaremos neste exemplo e inserir o primeiro registro no caixa.

```
CREATE TABLE CAIXA
(
    DATA            DATETIME,
    SALDO_INICIAL   DECIMAL(10,2),
    SALDO_FINAL     DECIMAL(10,2)
)
GO

INSERT INTO CAIXA
VALUES (CONVERT(DATETIME, CONVERT(VARCHAR, GETDATE(), 103)), 100, 100)
GO

CREATE TABLE VENDAS
(
    DATA    DATETIME,
    CODIGO  INT,
    VALOR   DECIMAL(10,2)
)
GO
```

Por lógica, o saldo final do caixa começa igual ao saldo inicial. Criemos então o primeiro trigger sobre a tabela de vendas, que aumentará o saldo final do caixa na data da venda quando uma venda for inserida.

```
CREATE TRIGGER TGR_VENDAS_AI
ON VENDAS
FOR INSERT
AS
BEGIN
    DECLARE
    @VALOR  DECIMAL(10,2),
    @DATA   DATETIME

    SELECT @DATA = DATA, @VALOR = VALOR FROM INSERTED

    UPDATE CAIXA SET SALDO_FINAL = SALDO_FINAL + @VALOR
    WHERE DATA = @DATA
END
GO
```

Nesse trigger utilizamos uma tabela temporária chamada INSERTED. Essa tabela existe somente dentro do trigger e possui apenas uma linha, contendo os dados do registro que acabou de ser incluído. Assim, fazemos um select sobre essa tabela e passamos o valores de suas colunas para duas variáveis internas, @VALOR e @DATA, que são utilizadas posteriormente para realizar o update na tabela de caixa.

O que fazemos é atualizar o saldo final da tabela caixa, somando o valor da venda cadastrada, no registro cuja data seja igual à data da venda (lógica de negócio simples).

Como sabemos que o saldo final da tabela caixa encontra-se com o valor 100,00, podemos testar o trigger inserindo um registro na tabela vendas. Vamos então executar a seguinte instrução SQL e observar seu resultado.

Momento Hands On

Continuando o script de banco anterior, crie uma trigger que remove o valor do caixa quando uma venda é deletada da tabela, ou seja, toda vez que uma venda é excluída, o valor referente a esta venda deve ser subtraída do caixa.

Crie uma tabela chamada produtos. Insira alguns produtos a esta tabela. É importante que esta tabela contenha um campo chamado QtdeEmEstoque.

Crie uma tabela chamada vendas, fazendo relação com a tabela produtos. Toda vez que um novo registro for inserido na tabela de vendas (ou seja, qdo ocorre uma venda de produto) subtraia em produtos a quantidade vendida do estoque.

Dessa forma, a cada insert na tabela VENDAS dispara uma trigger que atualiza a tabela produtos

Junção de Tabelas

CROSS JOIN

Quando queremos juntar duas ou mais tabelas por cruzamento. Ou seja, para cada linha da tabela FUNCIONARIO queremos todos os CARGOS ou vice-versa.

```
select funcionarios.nome,cargos.nomeCargo
from funcionarios cross join cargos
```

INNER JOIN

Quando queremos juntar duas ou mais tabelas por coincidência. Para cada linha da tabela FUNCINARIO queremos o CARGO correspondente que internamente (INNER), em seus valores de atributos, coincidam. No caso de FUNIONÁRIO e CARGO os atributos internos coincidentes são codigoCargo na tabela CARGO e codigoCargo na tabela FUNCIONARIO.

Note que idCargo é chave primária da tabela CARGO e chave estrangeira na tabela FUNCIONARIO. Para efetivarmos a junção das duas tabelas se fará necessário ligar (ON) as duas tabelas por seus atributos internos (INNER) coincidentes.

```
select funcionarios.nome,cargos.nomeCargo
from funcionarios inner join cargos on funcionarios.idCargo=cargos.idCargo
```

LEFT OUTER JOIN

- Se desejarmos listar todos os funcionários com seus respectivos cargos, incluindo os funcionários sem cargospoderíamos usar todo o poder da junção INNER JOIN adicionando ainda OUTER (EXTERNOS/OUTROS) Funcionários que não fazem parte do INNER JOIN, justamente àqueles sem cargos.

  Podemos conseguir esse feito com a junção FUNCIONARIO / CARGO através da declaração FUNCIONARIO OUTER LEFT JOIN CARGO, que promove a junção interna (INNER) de todos os funcionários a cargos e lista ainda outros (EXTERNOS/OUTER) não associados.

```
select funcionarios.nome,cargos.nomeCargo
from funcionarios left outer join cargos on funcionarios.idCargo=cargos.idCargo
```

RIGHT OUTER JOIN

Se desejarmos listar todos os CARGOS e seus respectivos FUNCIONARIOS, incluindo os CARGOS sem FUNCIONÁRIOS, poderíamos usar a junção RIGTH OUTER JOIN

```
select funcionarios.nome,cargos.nomeCargo
from funcionarios right outer join cargos on funcionarios.idCargo=cargos.idCargo
```

FULL OUTER JOIN

Aqui juntamos o poder das junções (JOIN) internas(INNER), a listagem de todas as outras linhas não associadas, tanto do lado direito (RIGHT) da junção como do lado esquerdo (LEFT).

```
select funcionarios.nome,cargos.nomeCargo
from funcionarios full outer join cargos on funcionarios.idCargo=cargos.idCargo
```

Momento Hands On

Imagine um site onde os jogadores podem criar seus personagens e conquistar itens para incrementar estes personagens. Baseado nisto, crie as tabelas capazes de armazenar estas informações.
Cadastre alguns personagens e itens.
Relacione personagens a itens e crie os selects:

- Selecione todos os personagens e seus itens
- Selecione um personagem específico e seus itens
- Selecione um item e todos os personagens que o possua.

O Departamento de Trânsito de sua cidade precisa de um sistema que controle as multas aplicadas aos veículos. Crie um banco de dados capaz de armazenar os tipos de multas, veículos e seus proprietários bem como as infrações aplicadas aos veículos.

Crie também os selects:

- Ver todas as multas aplicadas a um veículo qualquer;
- Ver todos os veículos que foram autuados por estacionamento proibido (ou outra infração a sua escolha).

OBRIGADO

profalex.deus@fiap.com.br

linkedin.com/in/alexsanderresende

FIAP MBA+


FIAP