Cursores

Cursores são estruturas da linguagem T-SQL que permitem o processamento das linhas retornadas por uma consulta (SELECT), através de estruturas de programação, como repetições ou comandos condicionais, ou seja, é a estrutura que permite que o resultado de uma instrução SELECT seja acessado linha a linha através de um laço While e alguns comandos específicos para cursores e é utilizado normalmente em conjunto com stored procedures.

Para melhor compreender o conceito de cursor, vamos considerar as seguintes tabelas

CLIENTES				
PK	**FK**	**Campo**	**Tipo**	**Null**
X		id	int identity	não
		Nome	varChar(40)	não

PRODUTOS				
PK	**FK**	**Campo**	**Tipo**	**Null**
X		id	int identity	não
		Descricao	varChar(40)	não
		Estoque	int	não
		ValorUnitario	money	não

VENDAS				
PK	**FK**	**Campo**	**Tipo**	**Null**
X		id	int identity	não
		Data	smalldatetime	não
	Clientes(id)	idCliente	int	não
		Situacao	bit	não

ITENS				
PK	**FK**	**Campo**	**Tipo**	**Null**
X	Vendas(id)	idVenda	int	não
X	Produtos(id)	idProd	int	não
		Quantidade	int	não
		Valor	money	não

Imagine que segundo esta estrutura de tabelas a venda será gravada juntamente com os seus itens, mas seu estoque não será baixado enquanto a venda não for finalizada.

Criaremos uma Stored Procedure de nome **sp_finaliza_vendas**, que alterará o campo situação da venda de 0 (aberta) para 1 (finalizada) e neste momento serão baixados todos os produtos do estoque de acordo com o que foi inserido na tabela itens.

Para iniciar a utilização de um cursor é necessário definir a instrução SELECT que ele acessará. No nosso exemplo, o nosso cursor vai manipular todos os itens da venda que deve ser finalizada conforme abaixo:

```
-- Select utilizado para criar o cursor
SELECT idProd, quantidade
    FROM Itens
    WHERE idVenda = @VenId
```

É interessante que você teste o comando select antes da criação do Cursor verificando se o resultado que ele apresenta é o resultado desejado. Utilizamos então o comando DECLARE para declarar as variáveis que forem necessárias e o CURSOR.

A variável @VenId será um parâmetro do procedimento, portanto não necessita declaração.

Segue a declaração do cursor:

```
--Declarando cursor
DECLARE CurItens --Nome do cursor
CURSOR FOR
   -- Select utilizado para criar o cursor
   SELECT idProd, quantidade
        FROM Itens
        WHERE idVenda = @VenId
```

Depois de realizada a declaração do cursor é necessário abri-lo e recuperar o seu primeiro registro. Para isto serão declaradas variáveis que receberão o código do produto e a quantidade vendida, através do comando FETCH.

O comando FETCH NEXT traz a próxima linha do SELECT, contudo o comando FETCH pode ser usado em conjunto com outras cláusulas para outros comportamentos, como o FETCH PRIOR (Anterior), FETCH FIRST (Primeiro), FETCH LAST (Último), entre outros.

```
--Declarando variáveis
DECLARE @ProdCod INTEGER, @Qtd MONEY

--Abrindo cursor
OPEN CurItens

--Atribuindo valores do select nas variáveis
FETCH NEXT FROM CurItens INTO @ProdCod, @Qtd
```

Este comando, mais precisamente o FETCH, definiu apenas os valores da primeira linha de retorno, contudo queremos utilizar o cursor para acessar várias linhas.

Para isso faremos um laço do tipo WHILE em e usaremos na condição a variável global do SQL Server @ @FETCH_STATUS, que retorna 0 (zero)

caso o último comando de FETCH tenha sido executado com sucesso e tenha retornado dados e –1 caso não haja mais dados (EOF – fim de arquivo).

Por fim utilizamos os comandos CLOSE , para fechar o cursor e DEALLOCATE, para eliminá-lo da memória, pois caso o procedimento seja executado novamente, pode haver erro na declaração do cursor caso ele ainda exista.

```
--Iniciando laço
WHILE @@FETCH_STATUS = 0
BEGIN
   --Próxima linha do cursor
   FETCH NEXT FROM CurItens
                 INTO @ProdCod, @Qtd
END

--Fechando e desalocando cursor
CLOSE CurItens
DEALLOCATE CurItens
```

Agora que recuperamos o código do produto e a quantidade em estoque podemos baixar o estoque na tabela de produtos com o cuidado de não deixarmos os estoque negativo, o que poderia ser feito através de uma restrição de domínio (check constraint) na tabela de estoque. Neste exemplo, faremos esta validação utilizando uma verificação com um SELECT na tabela de estoque. Caso o estoque não fique negativo, o comando para baixar o estoque é realizado, caso contrário será indicado um erro com o comando RAISERROR

```
--Iniciando laço
WHILE @ @FETCH_STATUS = 0  -- unir os dois @s
BEGIN
   IF (SELECT estoque - @Qtd FROM Produtos WHERE id =
@ProdCod) >= 0

       UPDATE Produtos SET estoque = estoque - @Qtd WHERE
id = @ProdCod
   ELSE RAISERROR('Estoque insuficiente!', 15, 1) --Próxima
linha do cursor

   FETCH NEXT FROM CurItens INTO @ProdCod, @Qtd
END

--Fechando e desalocando cursor
CLOSE CurItens
DEALLOCATE CurItens
```

Para finalizar, caso tudo tenha ocorrido com sucesso, devemos finalizar a venda propriamente dita, mudando o campo Situacao de 0 para 1. Uma prática muitíssimo recomendada é trabalhar com **transação**, pois caso um item dê problemas, os demais que já teriam sido baixados devem ser retornados.

Desta forma o procedimento completo ficaria como descrito abaixo

```
--Procedimento para finalização de uma venda
CREATE PROCEDURE sp_finaliza_venda (@VenCod INTEGER) AS

   --Declarando cursor
   DECLARE CurItens --Nome do cursor
      CURSOR FOR
         -- Select utilizado para o cursor
         SELECT idProd, Quantidade FROM Itens WHERE idVenda
= @VenCod

   --Declarando variáveis
   DECLARE @ProdCod INTEGER, @Qtd MONEY

   --Iniciando transação
   BEGIN TRANSACTION

   --Abrindo cursor
   OPEN CurItens

   --Atribuindo valores do select nas variáveis
   FETCH NEXT FROM CurItens INTO @ProdCod, @Qtd

   --Iniciando laço
   WHILE @@FETCH_STATUS = 0
   BEGIN
      IF (SELECT estoque - @Qtd FROM Produtos WHERE id =
@ProdCod) >= 0
         UPDATE Produtos
             SET estoque = estoque - @Qtd
             WHERE id = @ProdCod
      ELSE
         BEGIN
            --Desfazendo o que foi realizado anteriormente
            ROLLBACK TRANSACTION

            --Levantando erro
            RAISERROR('Estoque insuficiente!', 15, 1)

           --Fechando e desalocando cursor aqui também,
             -- pois o return sairá do procedimento
            CLOSE CurItens
            DEALLOCATE CurItens

           --Saindo do procedimento
            RETURN
         END -- else

      --Próxima linha do cursor
      FETCH NEXT FROM CurItens INTO @ProdCod, @Qtd
```

```
    END -- while

    --Fechando e desalocando cursor
    CLOSE CurItens
    DEALLOCATE CurItens

    --Caso tudo tenha ocorrido OK, alterando a situação da
venda
    UPDATE Vendas SET situacao = 1
       WHERE id = @VenCod

    --Confirmando transação
    COMMIT TRANSACTION
```

Algumas considerações sobre cursores:

- Um cursor deve estar sempre associado a uma consulta, especificada ao declarar o cursor.
- O comando FETCH popula as variáveis recebidas como parâmetro com os valores da próxima linha da consulta a ser lida. O número de variáveis passadas como parâmetro deve ser igual ao número de colunas retornadas na consulta associada ao cursor.
- A variável global @ @FETCH_STATUS retorna o resultado da última operação FETCH executada por um cursor na conexão.

  O status 0 significa que o comando FETCH retornou uma linha, qualquer outro resultado significa que não houve linha retornada.

- Cursores são estruturas relativamente lentas se comparadas ao desempenho de consultas do banco. O uso descuidado dessa ferramenta pode causar sérios problemas de performance.
- O uso do comando CLOSE ainda permite que o cursor seja aberto novamente.
- Sempre use o comando DEALLOCATE para destruir o cursor não permitindo mais a utilização do comando OPEN.

Fontes: http://imasters.com.br/artigo/3650/sql_server/utilizando_cursores/
http://www.mssqltips.com/sqlservertip/1599/sql-server-cursor-example/