



Log In / Cadastre-se

Pesquisar

[HOME](#) [DESENVOLVIMENTO](#) [FRONT-END](#) [BANCO DE DADOS](#) [EM DESTAQUE](#) [TODOS](#)[PUBLIQUE](#)

Desenvolvimento - C#

Guia prático para o desenvolvimento de Aplicações C# em Camadas - parte 2

Este é o segundo artigo da série onde vamos demonstrar passo-a-passo a construção de uma aplicação .Net utilizando o conceito de desenvolvimento em camadas.

por Carlos Camacho



g+1

1

2 – Criando a infra-estrutura de Banco de dados

Para o nosso projeto vamos precisar de uma infra-estrutura simples de banco de dados com apenas três tabelas: clientes, produtos e vendas.

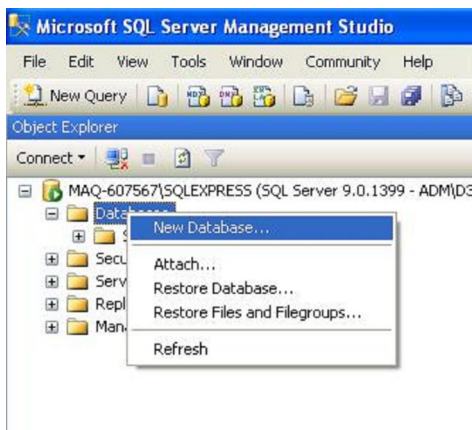
Se você estiver utilizando o MS SQL Server 2005, poderá abrir o Microsoft SQL Server Management Studio.

Selecione o servidor de banco de dados e o usuário para realizar a conexão como a seguir:



Estabelecida a conexão, podemos criar um banco de dados para o nosso projeto.

Para isso, clique com o botão direito sobre Databases e escolha a opção “New Database...”



Vamos dar o nome do nosso database de Loja.

Deixe a configuração padrão para o tamanho inicial dos arquivos de dados e Log. Clique em **Ok** para criar o database:



Publicidade

Compuware dynaTrace
É muito simples monitorar **100%** das transações do seu App



Experimente a Versão **FREE TRIAL**

Compuware **APM**

REVISTAS DEVMEDIA



.net Mag 116



Easy .net mag 37

VER TODAS

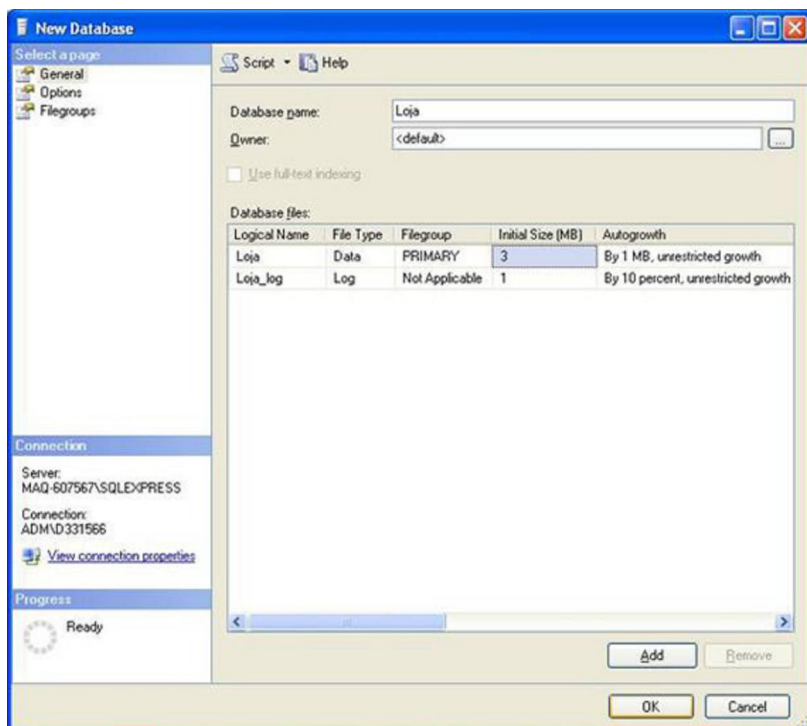
ASSINE

TOP 10 - ARTIGOS

TOP 10 - AUTORES

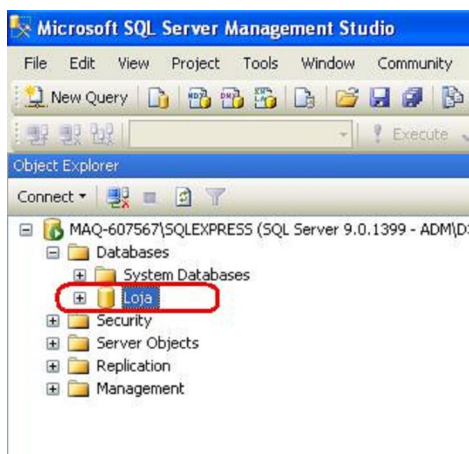
- 1 HTML Básico
- 2 Menu em CSS - Menu dropdown horizontal com HTML5 e CSS3
- 3 Comandos básicos em SQL - insert, update, delete e select
- 4 Criando um sistema de cadastro e login com PHP e MySQL
- 5 Código para background HTML e CSS
- 6 Copiando dados com o Robocopy
- 7 Trabalhando com Div em HTML
- 8 Excel: Como verificar se existe valores duplicados
- 9 Botão com CSS 3: Como criar um botão sem imagens
- 10 HTML Avançado

VER TODOS



Após clicar em Ok provavelmente precisaremos aguardar alguns segundos para que o MS SQL Server 2005 crie o banco de dados Loja.

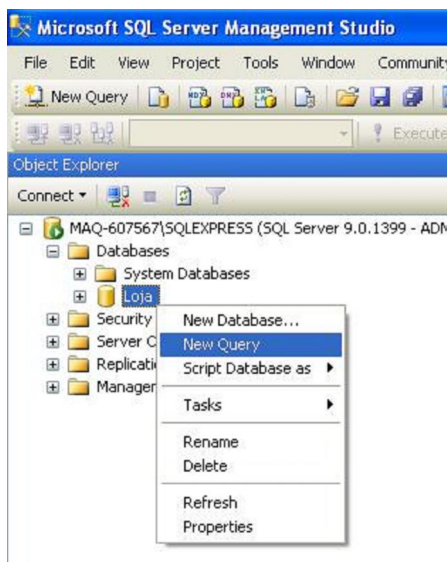
Quando esta janela de título *New Database* desaparecer, podemos verificar que o banco de dados Loja foi criado com sucesso conforme figura abaixo:



Agora que já temos o nosso banco de dados, vamos executar os scripts para criar as três tabelas necessárias ao nosso projeto.

2.1) Criando a tabela de Clientes

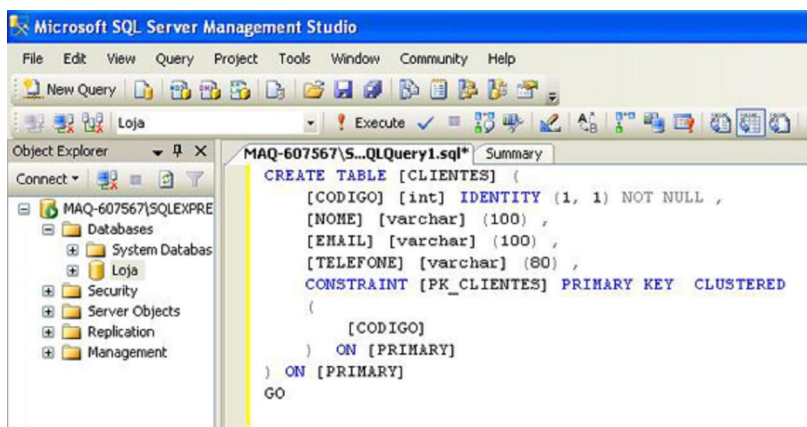
Clicando com o botão direito sobre o database Loja, escolha a opção **New Query**:



Copie o script SQL abaixo e cole na janela de Query aberta.

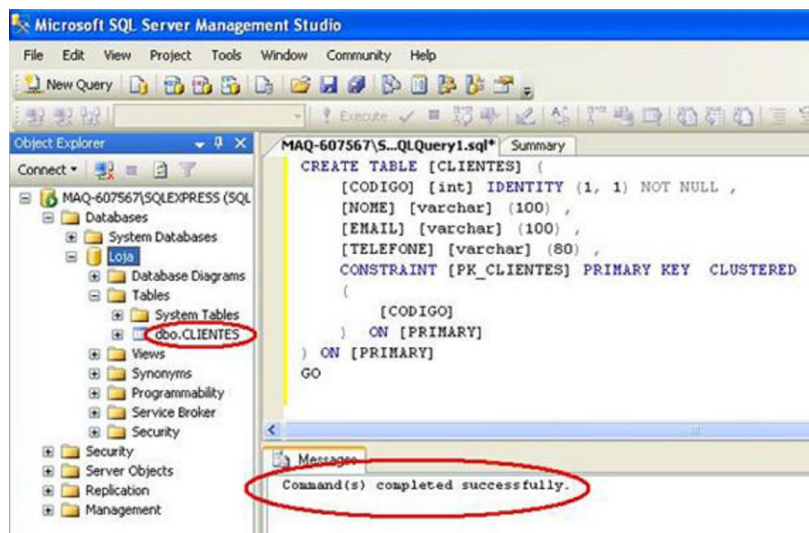
```
CREATE TABLE [CLIENTES] (  
[CODIGO] [int] IDENTITY (1, 1) NOT NULL ,  
[NOME] [varchar] (100) ,  
[EMAIL] [varchar] (100) ,  
[TELEFONE] [varchar] (80) ,  
CONSTRAINT [PK_CLIENTES] PRIMARY KEY CLUSTERED  
(  
[CODIGO]  
) ON [PRIMARY]  
) ON [PRIMARY]  
GO
```

O código deverá ficar assim:



Para executar esse script e criar a tabela de Clientes, clique no botão **Execute**:

Se a mensagem "*Command(s) completed successfully.*" for exibida, podemos expandir o database *Loja* e depois expandir *Tables* para vermos a tabela *CLIENTES* que acabamos de criar.



O campo *Código* da nossa tabela de Clientes foi criado como *Identity* e *Primary Key*. O que significa isso?

Bem, *Primary key* (chave primária) é um campo que será único para cada registro da tabela, ou seja, só existirá um campo com o código igual a 1, só existirá um campo com o código igual a 2 etc. É um campo que não admite valores repetidos. Se tentarmos incluir um valor na chave primária já existente na tabela, o MS SQL Server vai exibir uma mensagem de erro e não permitirá essa inclusão.

A característica *Identity* desse campo significa que ele será gerenciado pelo MS SQL Server e nós não precisamos nos preocupar em inserir um valor inexistente na tabela para evitar a duplicidade. Na prática isso quer dizer que nós nem precisamos mencionar esse campo nas operações de inserção pois é o gerenciador de banco de dados que se encarrega de inserir uma chave primária válida.

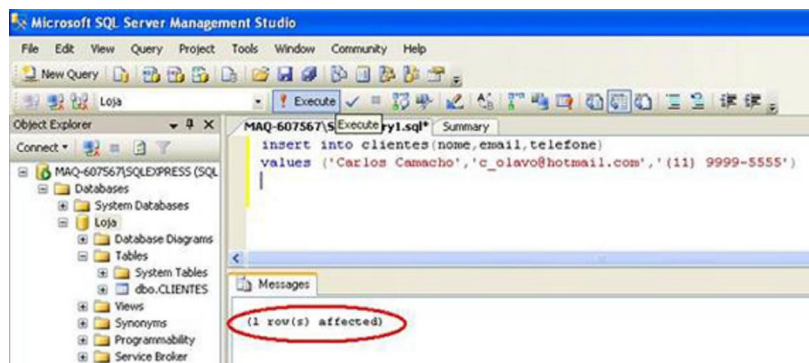
Vamos inserir um registro na tabela de clientes com o comando a seguir.

Podemos apagar o comando de Create table existente na janela do Management Studio e vamos colar o comando abaixo:

```
insert into clientes(nome,email,telefone)
```

```
values ('Carlos Camacho','c_olavo@hotmail.com','(11) 9999-5555')
```

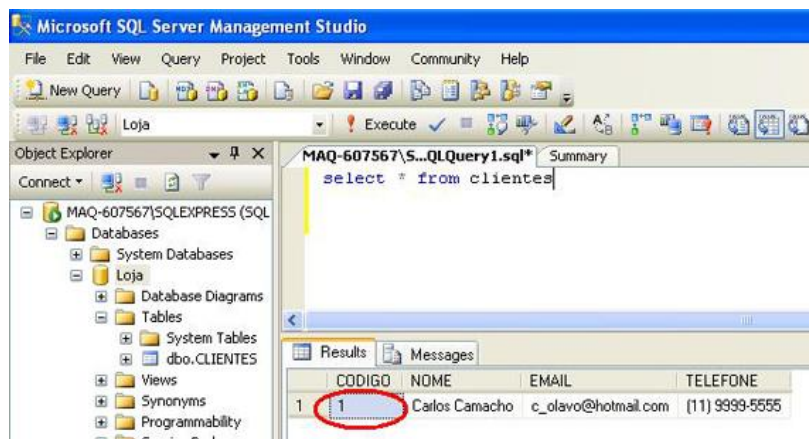
Clique em **Execute** para executar o comando de inserção:



Se a mensagem "1 row(s) affected" for exibida, significa que o registro foi inserido com sucesso.

Execute o comando de seleção a seguir para visualizarmos o registro que acabamos de inserir na tabela de Clientes:

```
Select * from clientes
```



Perceba que apesar de não mencionarmos o campo *Codigo* no comando de *insert* executado anteriormente, o MS SQL Server providenciou a inserção desse valor automaticamente.

Se executarmos outro comando de *insert*, o MS SQL Server vai gerar um código diferente para inserir no campo *Codigo* do novo registro.

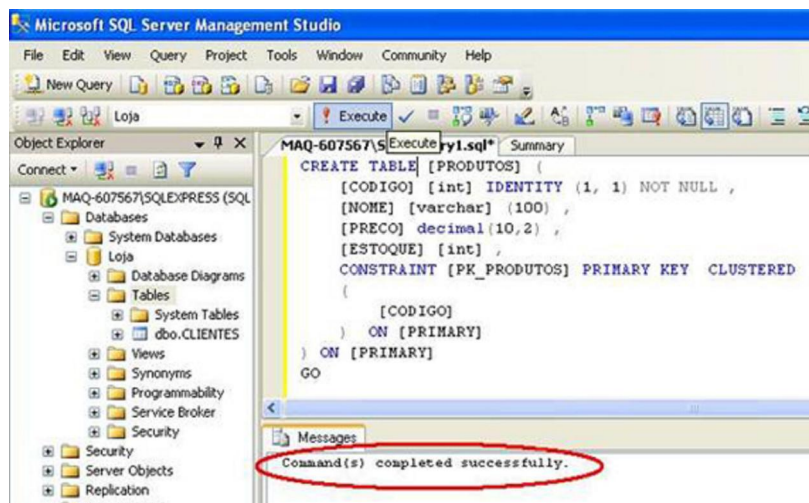
Parabéns! Acabamos de preparar a infra-estrutura do nosso projeto com a tabela de Clientes criada corretamente.

2.2) Agora vamos executar o script para criar a tabela de produtos:

O processo é o mesmo. Apague o conteúdo da janela de query e cole o script abaixo.

```
CREATE TABLE [PRODUTOS] (  
[CODIGO] [int] IDENTITY (1, 1) NOT NULL ,  
[NOME] [varchar] (100) ,  
[PRECO] decimal(10,2) ,  
[ESTOQUE] [int] ,  
CONSTRAINT [PK_PRODUTOS] PRIMARY KEY CLUSTERED  
(  
[CODIGO]  
) ON [PRIMARY]  
) ON [PRIMARY]  
GO
```

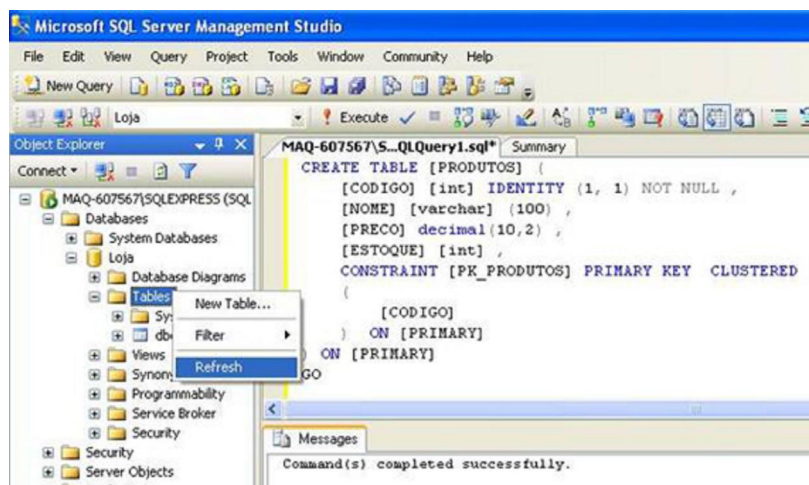
Com o script na janela de query, clique em **Executar** para criarmos a tabela de produtos.



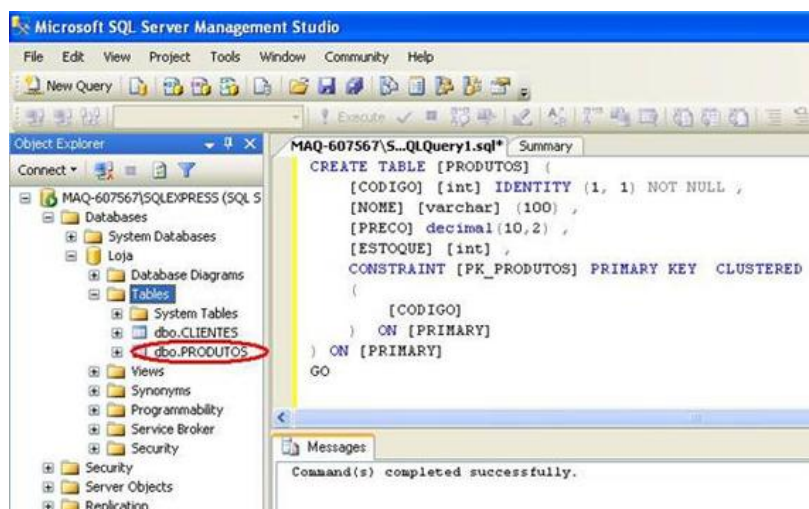
Nesse instante você deve estar pensando...

Ué?! Depois que cliquei no **Execute** apareceu a mensagem "Command(s) completed successfully." Mas a tabela PRODUTOS não apareceu em Tables... O que aconteceu?

Você está certo(a)! Precisamos atualizar o Management Studio para ver a tabela recém-criada. Para isso, vamos clicar com o botão direito do mouse em Tables e escolher a opção **Refresh**:



Veja o Management Studio após o Refresh:



Agora podemos ver a tabela de Produtos.

Que tal inserirmos dois produtos nessa tabela?

- Apague o código da janela de query;
- Copie o código abaixo na janela de query e clique em **Execute** para a inserção dos dois produtos.

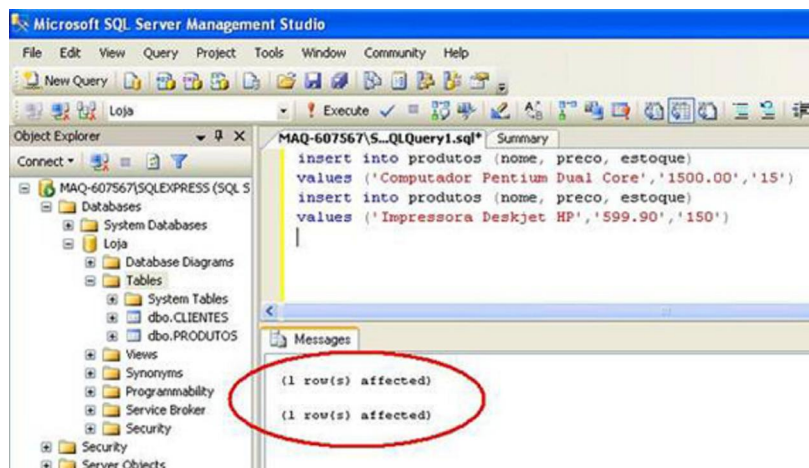
```
insert into produtos (nome, preco, estoque)
```

```
values ("Computador Pentium Dual Core","1500.00","15")
```

```
insert into produtos (nome, preco, estoque)
```

```
values ("Impressora Deskjet HP","599.90","150")
```

Vamos ver o resultado:



Na janela de mensagens podemos ver que os dois registros foram inseridos com sucesso.

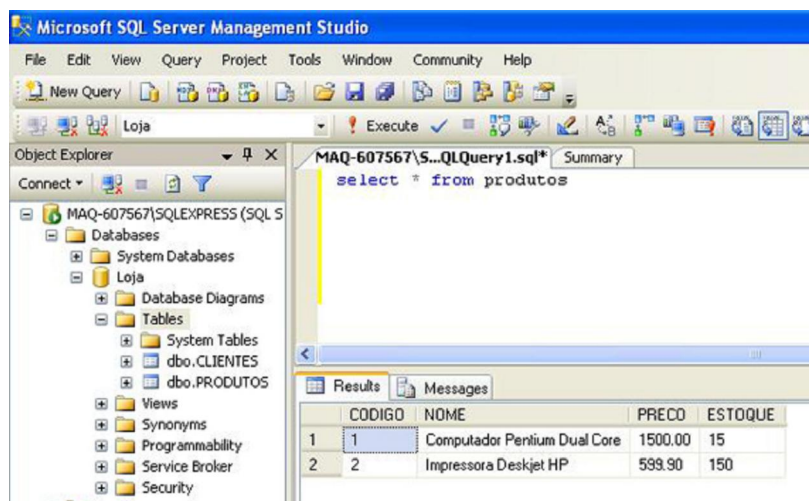
Utilize o comando select para consultar os produtos cadastrados.

- Apague o conteúdo da janela de query;

- Copie o comando abaixo e clique em **Execute**.

```
select * from produtos
```

O resultado é esse:



Ok, a nossa inserção de produtos funcionou!

Cadastramos 15 computadores Pentium Dual Core com o valor unitário de R\$ 1.500,00;

E cadastramos também 150 impressoras Deskjet HP com o valor unitário de R\$ 599,90.

A nossa tabela de Produtos está pronta. Agora só falta criarmos a tabela de Vendas!

2.3) Vamos criar a tabela para armazenar as Vendas.

- Apague o conteúdo da janela de query;

- Copie o comando abaixo e clique em **Execute**.

```
CREATE TABLE [VENDAS] (
```

```
[CODIGO] [int] IDENTITY (1, 1) NOT NULL ,
```

```
[DATA] [datetime],
```

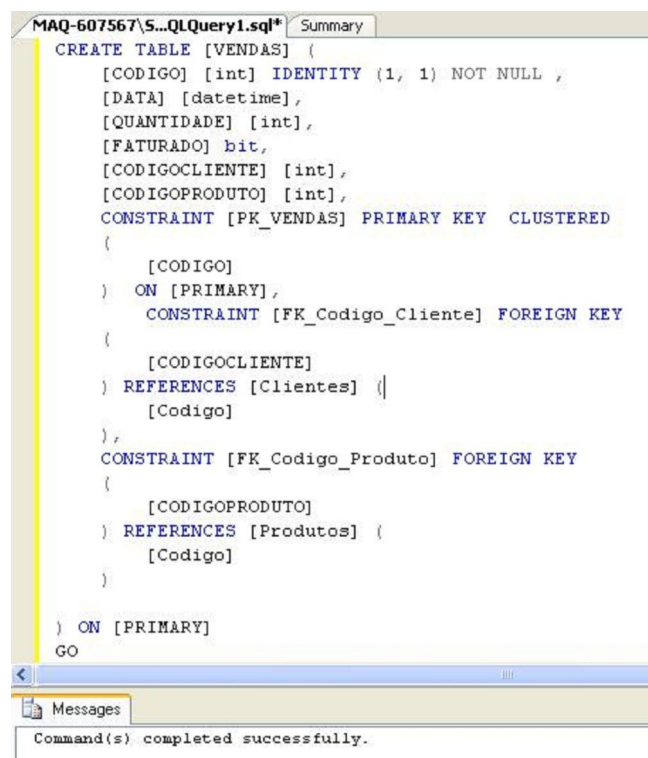
```
[QUANTIDADE] [int],
```

```
[FATURADO] bit,
```

```
[CODIGOCLIENTE] [int],
```

```
[CODIGOPRODUTO] [int],  
  
CONSTRAINT [PK_VENDAS] PRIMARY KEY CLUSTERED  
  
(  
  
[CODIGO]  
  
) ON [PRIMARY],  
  
CONSTRAINT [FK_Codigo_Cliente] FOREIGN KEY  
  
(  
  
[CODIGOCLIENTE]  
  
) REFERENCES [Clientes] (  
  
[Codigo]  
  
),  
  
CONSTRAINT [FK_Codigo_Produto] FOREIGN KEY  
  
(  
  
[CODIGOPRODUTO]  
  
) REFERENCES [Produtos] (  
  
[Codigo]  
  
)  
  
) ON [PRIMARY]  
  
GO
```

Vejam os resultados:



```
MAQ-607567\S...QLQuery1.sql* Summary  
CREATE TABLE [VENDAS] (  
    [CODIGO] [int] IDENTITY (1, 1) NOT NULL ,  
    [DATA] [datetime],  
    [QUANTIDADE] [int],  
    [FATURADO] bit,  
    [CODIGOCLIENTE] [int],  
    [CODIGOPRODUTO] [int],  
    CONSTRAINT [PK_VENDAS] PRIMARY KEY CLUSTERED  
    (  
        [CODIGO]  
    ) ON [PRIMARY],  
    CONSTRAINT [FK_Codigo_Cliente] FOREIGN KEY  
    (  
        [CODIGOCLIENTE]  
    ) REFERENCES [Clientes] (  
        [Codigo]  
    ),  
    CONSTRAINT [FK_Codigo_Produto] FOREIGN KEY  
    (  
        [CODIGOPRODUTO]  
    ) REFERENCES [Produtos] (  
        [Codigo]  
    )  
    ) ON [PRIMARY]  
GO  
  
Messages  
Command(s) completed successfully.
```

Dê um Refresh clicando com o botão direito em Tables para que a tabela de Vendas seja exibida:



A tabela de Vendas que acabamos de criar possui duas *foreign keys* (chaves estrangeiras). Essas duas chaves são os campos: *CodigoCliente* e *CodigoProduto*.

Isso significa que quando um registro for inserido nesta tabela, o campo *CodigoCliente* incluído deverá existir na tabela de Clientes, assim como o campo *CodigoProduto* deverá existir na tabela Produtos.

Se alguém tentar incluir valores nestes campos que não existam nas tabelas citadas o Microsoft SQL Server vai informar a ocorrência de um erro e não permitirá a inclusão.

Esse mecanismo do sistema gerenciador de banco de dados existe para manter a integridade dos dados. Chamamos isso de *integridade referencial*.

Uma regra que vamos usar no nosso projeto é que ao realizar uma venda, atualizaremos o estoque do produto na tabela de Produtos. Faremos com que o estoque fique atualizado com a seguinte fórmula:

$$\text{Estoque} = \text{Estoque} - \text{Quantidade Vendida} .$$

Essa tarefa será realizada juntamente com a tarefa de inclusão da Venda na tabela de Vendas.

Assim, para manter a integridade dos dados, precisaremos garantir que as duas coisas aconteçam:

- 1) a inclusão na tabela de Vendas; e
- 2) a atualização do estoque na tabela de Produtos.

Para isso usaremos uma *Transação*. Com a transação podemos garantir a execução das duas tarefas. Se uma delas for bem sucedida e a outra falhar, a transação desfaz a primeira tarefa mantendo assim a integridade referencial.

Em outras palavras, não existirá na tabela de Vendas um registro que tenha o código do produto vendido mas que o estoque deste produto na tabela de Produtos esteja desatualizado.

Veremos isso com mais detalhes no próximo artigo chamado “Camada de Acesso a Dados (Data Access Layer)”. Implementaremos as classes de acesso a dados e faremos esse controle transacional no método Incluir da classe de Vendas.

Observações:

Para futura utilização no nosso projeto, precisaremos guardar as seguintes informações sobre o banco de dados:

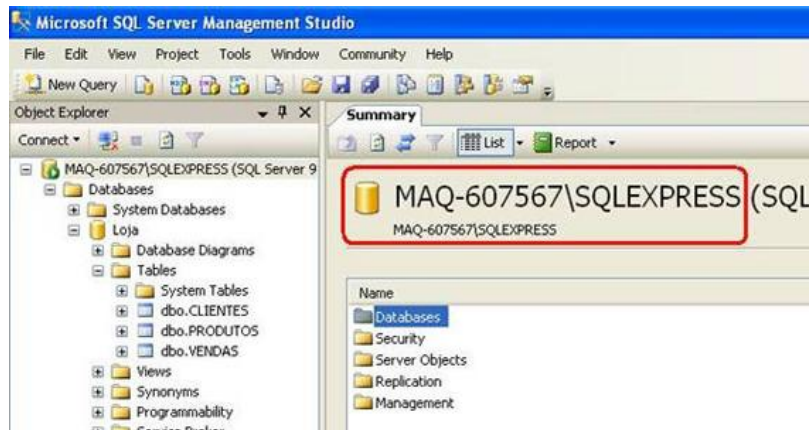
- Nome do servidor de banco de dados que estamos usando;
- Nome do banco de dados onde as três tabelas foram criadas;
- Nome do usuário e senha utilizados para acessar o banco de dados.

Essa informação é importante para a construção da nossa string de conexão ou *connectionstring*.

A *connectionstring* será usada para a conexão com o banco de dados através da aplicação que desenvolveremos neste projeto.

Um bom site para consulta sobre a correta *connectionstring* que devemos utilizar é o <http://www.connectionstrings.com/> Neste site existem dicas para criar *connectionstrings* para diferentes versões de bancos de dados.

No meu caso, se eu fechar a janela de Query que estávamos usando, poderei ver o nome do servidor de banco de dados que estou usando:



O usuário e senha são aqueles que usei inicialmente para conectar no servidor de banco de dados antes de iniciar a criação do database Loja.

Nome do Servidor: *MAQ-607567\SQLEXPRESS*

Nome do banco de dados: *Loja*

Usuário: *camacho*

Senha: *camacho2008a3*

Assim, a minha *connectionstring* ficará assim:

```
"server=MAQ-607567\SQLEXPRESS;database=Loja;user=camacho;pwd=camacho2008a3"
```

A sua *connectionstring* vai depender dos nomes que você usou para:

- Criar o servidor de banco de dados;
- Criar o banco de dados;
- Criar usuário e senha que terão acesso ao banco de dados.

Até o próximo artigo!



Carlos Camacho - Carlos Olavo de Azevedo Camacho Júnior é mestrando em Tecnologias da Inteligência e Design Digital pela Pontifícia Universidade Católica de São Paulo PUCSP. Pós-graduado em Análise e Projeto de Sistemas pela Universidade Paulista UNIP. Bacharel em Ciência da Computação pela Universidade Paulista UNIP e possui Licenciatura Plena em Matemática pelas Faculdades Oswaldo Cruz.

MCP .Net, MCP SQL Server, Carlos Camacho leciona disciplinas técnicas na área de Ciências Exatas e é Consultor em Tecnologia da Informação para Instituições Financeiras.

Lambda Expressions x SQL: Comparando a sintaxe de consultas comuns
C#

List: trabalhando com listas genéricas em C#
C#

Criando Gráficos usando C# e API do Google
C#


Imprimindo um Panel com C#
C#

Consumindo um Web API em C#
C#

Estamos aqui:    











Linha de Código faz parte do grupo Web-03

[Política de privacidade e de uso](#) | [Anuncie](#) | [Cadastre-se](#) | [Fale conosco](#)

 **Linha de Código**

Like

9,981 people like Linha de Código.



Facebook social plugin

© 2014 Linha de Código. Todos os direitos reservados