easy .net Magazine 29 - Índice

Conhecendo a arquitetura ADO.NET - Revista easy .net Magazine 29

Este artigo apresenta o conceito, a arquitetura, as principais interfaces e providers da tecnologia ADO.NET, apresentando as principais classes e métodos que podemos utilizar para interagir com bases de dados nos nossos sistemas.

Artigo do tipo Tutorial

Autores: Giuvane Conti e Everton Coimbra de Araújo

Conhecendo a arquitetura ADO.NET

Toda plataforma de desenvolvimento possui uma API específica para ligar com o acesso a dados, no caso do .NET esta API é o ADO.NET que encapsula todas as classes necessárias para realizarmos a comunicação de nossa aplicação com uma base de dados nos fornecendo mecanismos de conexão com a mesma além da possibilidade de execução de operações como inserção, atualização, deleção e consulta de dados através de instruções SQL.

Este artigo apresenta o conceito, a arquitetura, as principais interfaces e providers da tecnologia ADO.NET, apresentando as principais classes e métodos que podemos utilizar para interagir com bases de dados nos nossos sistemas. Além disso, apresentamos ainda a conexão do ADO.Net com o SQL Server e uma implementação do pattern DAO. Na parte prática teremos a implementado um exemplo simples envolvendo Pedido e Itens de Pedido utilizando um DAO com classes ADO.Net para realizar as operações na base de dados.

Em que situação o tema é útil

O ADO.NET é útil em qualquer sistema que necessite de uma conexão com um banco de dados para manipulação de dados como inserir, editar, deletar e buscar, ou seja, operações CRUD. Além disso, o uso direto das classes do ADO.NET é mais rápido do que o uso de frameworks ORM, fazendo com que em aplicações críticas o mesmo possa se tornar um diferencial para a performance da mesma.

ADO.NET é a biblioteca utilizada pela Microsoft para acesso a bases de dados na plataforma .NET, podendo ser utilizado com qualquer linguagem de programação que

tenha acesso a esta plataforma.

No pacote de tecnologias do ADO.NET existem classes que auxiliam a aplicação deste em projetos que utilizem a plataforma .NET e dentro deste contexto alguns dos recursos que se destacam são as interfaces disponíveis, suas respectivas implementações concretas e seus providers, que são os provedores específicos para cada fonte de dados que pode ser utilizada, por exemplo, o ADO.NET disponibiliza providers para conexão com SQL Server, MySQL, FireBird, Oracle, Sybase, Access, XML, dentre outros.

Existem algumas bibliotecas e recursos que realizam as operações disponíveis no ADO.NET de forma mais fácil tirando proveito de todo o conteúdo disponível pelo ADO.NET e facilitando a vida do programador aumentando a produtividade, como é o caso da biblioteca Entity Framework, que é parte integrante do pacote ADO.NET e do conjunto de recursos oferecidos pelo LINQ. Além destas tecnologias, ainda é possível implementar um conceito de programação chamado DAO Genérico, baseado nestas tecnologias, onde as operações CRUD (Create, Read, Update e Delete) podem ser aplicadas para qualquer classe que represente uma tabela do banco de dados dentro de um projeto, facilitando ainda mais a vida do programador.

Neste artigo será criado um exemplo prático, de controle de Pedido e Itens de Pedido, utilizando o conceito de DAO sendo implementado com base nas classes do ADO.NET utilizando o provider de conexão com o SQL Server.

O que é o ADO.NET ?

O ADO.NET é uma evolução do ADO (ActiveX Data Objects) que foi utilizado até o Visual Basic 6, apesar de ter sido construído sem nenhum reaproveitamento da tecnologia ADO, a Microsoft manteve muitos dos conceitos do mesmo no ADO.NET,

apesar destes possuírem uma arquitetura totalmente diferente.

O ADO.NET possui diversas características, como por exemplo: interoperabilidade através do uso de XML para intercambio de dados, performance, robustez, escalabilidade com modelo desconectado e produtividade.

Arquitetura, interfaces e classes concretas

A arquitetura do ADO.NET está dividida em dois grupos: Managed Providers (provedores gerenciados) e Content Components (componentes de conteúdo). Os componentes do grupo Managed Providers se encarregam do acesso a dados, porém não armazena os mesmos, fazendo ligação direta com a fonte de dados. Neste grupo temos classes de conexão, transação, execução de comandos (Command) e leitura de dados (DataReader).

Já o grupo de Content Components é responsável por armazenar e manipular os dados em memória, porém eles não sabem de sua origem e nem o que significam, sendo que neste grupo temos as classes DataSet, DataTable, DataRow, DataColumn, etc.

Na **Figura 1** é possível ver a representação da arquitetura ADO.NET, onde a parte referente ao DataSet representa os Content Components.

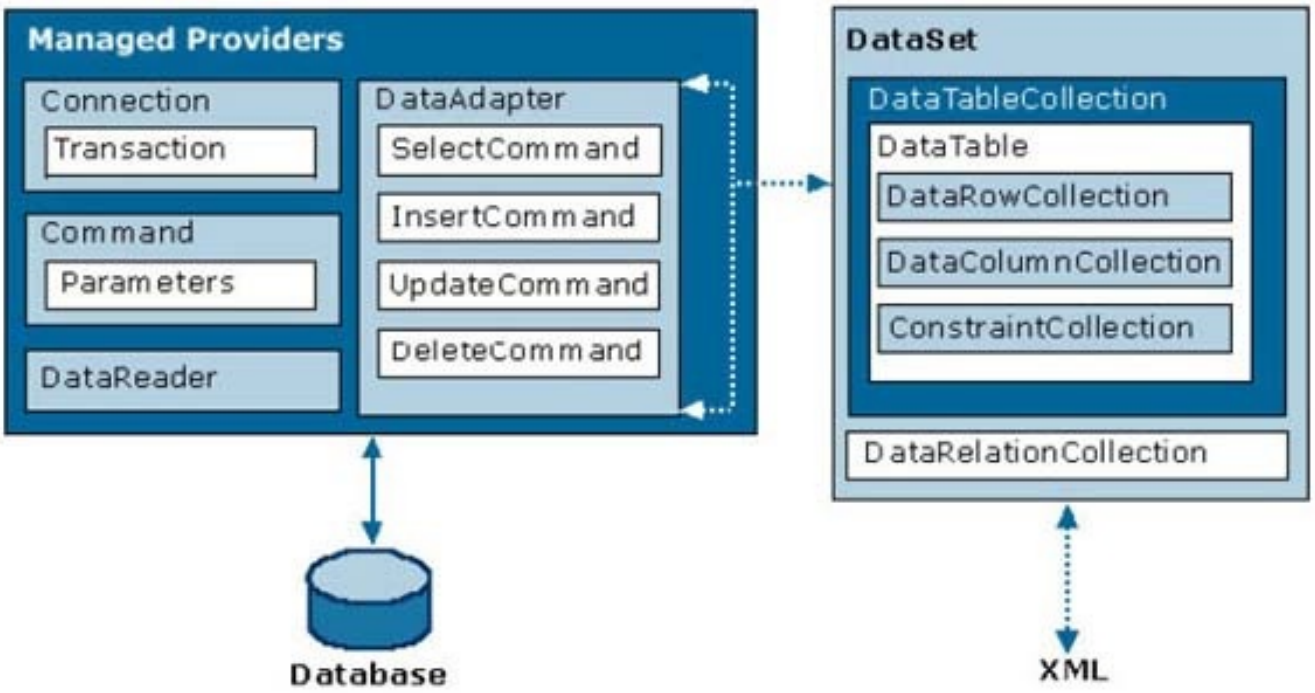


Figura 1. Arquitetura da tecnologia ADO.NET

O grupo Managed Providers possui um conjunto de classes concretas que possuem 5 interfaces principais sendo IDbConnection, IDbCommand e IDbDataReader, IDbDataTransaction e IDbDataAdapter. A partir destas interfaces qualquer fabricante de banco de dados pode criar componentes que implementam as mesmas para permitir o acesso ao seu SGBD, desta forma o ADO.NET pode ser estendido para diversos SGBDs, além de nos permitir construir uma arquitetura abstrata na nossa aplicação, visto que podemos fazer uso das interfaces como referência em nossos projetos. O conjunto de componentes que implementam estas interfaces são como Providers e serão abordados mais adiante neste artigo.

A interface **IDbConnection** define métodos e propriedades de uma fonte de dados, a propriedade mais importante é um string de conexão com o banco de dados onde são informados os detalhes da conexão como nome do servidor, nome do banco de dados, nome do usuário e senha. Além disso, ela possui os métodos Open() e Close(), que são responsáveis pelo controle de abertura e fechamento da conexão com a base de

dados, além disso as classes que implementam a interface IDbConnection trabalham em conjunto com um cache de conexões, permitindo que os métodos Open() e Close() sejam executados rapidamente.

A interface **IDbCommand** é responsável pela execução de comandos SQL e stored procedures do banco de dados, informando parâmetros quando necessário, além disso ela possui três formas de execução de comandos SQL:

· **ExecuteReader**: Executa um comando e retorna um cursor apenas de leitura para acessar o conjunto de resultados da consulta.

· **ExecuteNonQuery**: Utilizado em instruções SQL onde não existem resultados como resposta, como por exemplo insert, update e delete.

· **ExecuteEscalar**: Retorna um valor simples, como uma string ou um valor inteiro. Usado normalmente quando temos, como por exemplo, um select count(*) em alguma tabela.

A interface **IdataReader** é responsável pela manipulação/leitura de um cursor de dados retornado pelo método ExecuteDataReader.

A interface **IDbDataTransaction** é responsável pelo gerenciamento de transações no banco de dados e a interface **IdbDataAdapter** é responsável por métodos de obtenção e atualização de dados.

As classes concretas que implementam estas interfaces encontram-se dentro do namespace System.Data, por exemplo, o namespace System.Data.SqlClient contém classes concretas que implementam estas interfaces para o SQL Server.

O grupo Content Components é manipulado pela classe DataSet, que se encontra no namespace System.Data, sendo que esta classe é um dos maiores componentes da

arquitetura do ADO.NET. Um DataSet é baseado em XML, contém um conjunto de dados em cache que não estão conectados ao banco de dados, não depende da fonte de dados e pode armazenar dados de múltiplas tabelas que podem possuir relacionamentos. O DataSet fornece as principais funcionalidades para criação de aplicações com banco de dados desconectados, porém também suporta o modelo conectado, esta questão ainda será abordada em detalhes mais a frente neste artigo.

O objeto **Connection** é responsável por estabelecer uma ligação com a fonte de dados específica, este objeto contém as informações necessárias para realizar a conexão, como o provider, o caminho do banco de dados, usuário e a senha de acesso. Cada tipo de provedor (banco de dados) possui um objeto Connection diferente e este realiza a conexão através de uma connection string. Algumas das propriedades da classe Connection são:

· **ConnectionString** - Contém as informações necessárias para realizar a conexão com o banco de dados como o endereço do mesmo, o usuário, senha, dentre outras.

· **DataBase -** Nome de banco de dados após a conexão ser aberta.

· **DataSource -** Nome da instância do banco de dados.

· **State -** Retorna o estado atual da conexão, por exemplo: Broken, Closed, Connecting, Executing, Fetching e Open.

Na **Listagem 1** é criada uma Connection do tipo SqlConnection, para conexão com o banco de dados SQL Server.

Listagem 1. Exemplo de uso do Objeto SqlConnection

```
1 // Informações da string de conexão
2 string Server = "localhost";
3 string Username = "usuario";
```

```
4 string Password = "senha";
5 string Database = "banco de dados";
6
7 // Montando a string de conexão
8 string ConnectionString = "Data Source=" + Server + ";";
9 ConnectionString += "User ID=" + Username + ";";
10 ConnectionString += "Password=" + Password + ";";
11 ConnectionString += "Initial Catalog=" + Database;
12 //cria uma instância do objeto Connection em memória
13 SqlConnection SQLConnection = new SqlConnection();
14 try
15 {
16       SQLConnection.ConnectionString = ConnectionString;
17       SQLConnection.Open();
18       // Aqui ficariam as operações
19
20 }
21 catch (Exception Ex)
22 {
23                MessageBox.Show(Ex.Message);
24 }
```

Na **linha 13** é onde a SqlConnection é instanciada após todas as informações referentes ao banco de dados serem integradas a string de conexão. Na **linha 17** temos a abertura da conexão, feito isso o objeto SqlConnection está pronto para realizar operações diretas no banco de dados. Todas as operações referentes ao banco de dados estão dentro de um bloco **Try/Catch**, para informarmos possíveis erros ao usuário, caso estes ocorram.

O objeto **Command** é utilizado para a interação com um banco de dados, em um Command é possível especificar qual o tipo da operação que se deseja realizar em um banco de dados, como por exemplo, incluir, alterar ou excluir dados. Basicamente um Command é utilizado para executar uma instrução SQL. Para criar um objeto Command é necessário que o objeto Connection já tenha sido configurado.

Na **Listagem 2** temos a criação de um objeto do tipo Command onde o mesmo é

utilizado após um objeto Connection ser configurado, o Command utilizado é do tipo **SqlCommand**, para realizar operações em um banco de dados SQL Server.

Listagem 2. Objeto Command

```
01 string sql = null;
02 string connetionString = @"Data Source=.\SQL;Initial
          Catalog=ArtigoAdoNet;Integrated Security=true";
03
04 sql = "Select Count(*) from produtos";
05 cnn = new SqlConnection(connetionString);
06
07 try
08 {
09       cnn.Open();
10       cmd = new SqlCommand(sql, cnn);
11       Int32 contador = Convert.ToInt32(cmd.ExecuteScalar());
12       cmd.Dispose();
13        cnn.Close();
14        MessageBox.Show(" No. de linhas " + contador );
15 }
16 catch (Exception ex)
17 {
18        MessageBox.Show("Não foi possível abrir a conexão com o banco
                          de dados ! " + ex.Message.ToString());
19 }
```

Na **linha 10** o objeto SqlCommand foi instanciado passando como parâmetro a instrução SQL a ser executada e a conexão já em modo aberto. Na **linha 11** o atributo contador recebe o resultado da execução do código SQL, através do método ExecuteScalar.

O objeto **DataReader** é responsável por acessar as informação de uma fonte de dados de forma mais rápida, porém ele é utilizado apenas para leitura de dados e exige que a conexão esteja aberta enquanto estiver sendo utilizado. Duas de suas características são forward-only, ou seja, somente podemos navegar nos seus dados em uma única

direção e read-only, onde apenas podemos realizar leitura nos dados do mesmo. O resultado gerado por um DataReader é armazenado em forma de buffer no cliente até que o retorno seja requisitado através do método **Read()**.

A ordem de utilização de um DataReader pode ser descrita da seguinte forma: criar uma instância de um Command, criar um DataReader, utilizar o método ExecuteReader() do Command, obter o retorno da consulta usando o método Read().

Na **Listagem 3** o objeto DataReader é utilizado para leitura de dados, usando um SqlDataReader, que realiza leitura de dados em um banco de dados SQL Server.

Listagem 3. Objeto DataReader

```
01 string conString = "Data Source==.\SQL;Initial
            Catalog=ArtigoAdoNet;Persist Security Info=false;User
            ID=giuvane;Password=123456"
02 string consulta = "SELECT * FROM Produto";
03 SqlConnection conexao = new SqlConnection("conString");
04 SqlCommand comando = new SqlCommand(consulta, conexao);
05 SqlDataReader dr = null;
06 try
07 {
08     conexao.Open();
09     dr = comando.ExecuteReader();
10     while (dr.Read())
11     {
12         Console.WriteLine(dr.GetString(1));
13}
14 }
15 catch{
16     Console.WriteLine("Erro.");
17 }
18 finally
19 {
20     dr.Close();
21     conexao.Close();
22 }
```

Com a conexão já aberta na **linha 09** é possível ver o objeto Command invocando o método ExecuteReader(), tendo este o seu retorno atribuído a um objeto do tipo DataReader, na **linha 10** o buffer é lido através do método Read() do atributo dr (que é do tipo DataReader).

O grupo Content Components é representado pelo objeto **DataSet**, sendo que este representa uma cópia do banco de dados em memória, um DataSet é muito utilizado no modelo desconectado de uso do ADO.NET, que será abordado mais a frente no tópico “Modelo Conectado e Modelo Desconectado”. Um DataSet trabalha com um conjunto de dados em cache não conectados ao banco de dados, ele não depende da fonte de dados e pode armazenar dados de múltiplas tabelas. Abaixo temos os principais métodos do objeto DataSet:

· **AcceptChanges –** Método responsável por gravar as alterações realizadas no DataSet.

· **Clear** – Método responsável por remover todas as linhas das tabelas do DataSet.

· **Clone** - Método que realiza uma cópia da estrutura de um DataSet.

· **Copy** - Método que realiza uma cópia da estrutura e dos dados de um DataSet.

· **GetChanges** - Método que retorna uma cópia dos dados que foram alterados em um DataSet ou dos dados que foram aplicados em um filtro definido em DataRowState.

· **GetXml** - Método que retorna uma representação em XML dos dados do DataSet.

· **GetXmlSchema** - Método que retorna um XML da estrutura do DataSet.

· **HasChanges** - Método que retorna um boolean indicando se existe ou não alterações pendentes no DataSet.

· **Merge** - Método que mescla o DataSet com um provider.

· **ReadXml** - Método que carrega um esquema XML e seus dados para o DataSet.

· **ReadXmlSchema** - Método que carrega apenas um esquema XML para o DataSet.

· **Reset** - Método que reseta o DataSet para seu estado original.

· **WriteXml** - Método que escreve os dados e o esquema XML do DataSet para um arquivo ou buffer.

· **WriteXmlSchema** - Método que escreve apenas o esquema XML do DataSet para um arquivo ou buffer.

Além disso, o DataSet possui duas propriedades muito importantes e úteis, sendo elas:

· **Relations –** Propriedade contendo uma lista de relações armazenadas em um objeto DataRelationCollection, que relaciona tabelas através de chaves estrangeiras.

· **Tables** - Uma lista de tabelas dos dados atuais do DataSet.

O objeto **DataTable** representa uma ou mais tabelas de dados em memória, os objetos DataTable são armazenados dentro de um DataSet. Abaixo temos as principais propriedades de um objeto **DataTable**:

· **Columns -** Representa uma lista com as colunas da tabela, sendo estas colunas do tipo DataColumn.

· **Rows** - Representa as linhas da tabela, sendo estas linhas do tipo DataRow.

· **PrimaryKey** – Indica qual a chave primária da tabela.

· **TableName** – Indica o nome do objeto DataTable.

· **AcceptChanges** - Atualiza as alterações feitas no DataTable na fonte de dados.

· **NewRow** - Adiciona um novo objeto DataRow ao DataTable.

· **Copy** - Copia os dados do DataTable.

· **Clear** - Limpa os dados do DataTable.

· **RejectChanges** - Ignora todas as alterações feitas no DataTable.

Providers do ADO.NET

Os providers do ADO.NET são pacotes de classes que possibilitam a interação com uma fonte de dados específica, como por exemplo, com uma base SQL Server. Cada pacote possui um prefixo que indica qual a fonte de dados que este pacote suporta. Um provider possui objetos de Connection, Command, DataAdapter e DataReader. O fluxo para uso destes objetos começa com a criação de uma Connection fornecendo a string de conexão, logo cria-se o objeto Command fornecendo a instrução SQL que deverá ser executada, se este Command retornar informações que precisarão ser manipuladas, então será necessário criar um objeto DataAdapter e assim preencher um DataSet ou DataTable com estas informações. Os principais providers de conexão existentes são:

· **ODBC -** Provider geralmente utilizado em bancos de dados antigos que não possuem uma implementação própria para o ADO.NET mas que possuem um provider para o ODBC.

· **OleDb** - Provider utilizado para usarmos fontes de dados Access ou Excel.

· **Oracle**- Provider utilizado em banco de dados Oracle.

· **SQL Server -** Provider utilizado em banco de dados SQL Server.

Para acessar uma fonte de dados a primeira coisa a ser feita e a mais importante é configurar uma string de conexão para possibilitar que o provider do ADO.NET crie e mantenha contato com o banco de dados. Os parâmetros de uma string de conexão podem variar de acordo com o provider utilizado, no nosso caso estamos usando o SQL Server como exemplo, com isso teremos os seguintes parâmetros na string de conexão:

· **Data Source**: Indica o local onde se encontra o banco de dados.

· **Initial Catalog**: Indica o nome do banco de dados.

· **Integrated Security**: Indica se a autenticação será integrada ao login do Windows.

· **UID e PWD**: Se a autenticação não for integrada ao login do windows, é necessário informar um username (UID) e uma senha (PWD).

Na **Listagem 4** é possível visualizar uma string de conexão utilizando o objeto SqlConnection.

Listagem 4. Exemplo de string de conexão

```
string connectionString = @"Data Source=.\SQL;
Initial Catalog=ArtigoAdoNet;Integrated Security=True";
  SqlConnection conn = new SqlConnection(connectionString);
```

Vale lembrar que estes são apenas os principais providers de conexão, porém existem muitas outras classes distribuídas diretamente pela Microsoft ou pelos próprios desenvolvedores dos ODBCs. Na **Tabela 1** é possível visualizar o nome do namespace e das classes dos principais providers para o ADO.NET.

Componente	Namespace/Classe
Connection	SqlClient / SqlConnection
	OracleClient / OracleConnection
	OleDB / OleDbConnection
	Odbc / OdbcConnection
Command	SqlClient / SqlCommand
	OracleClient / OracleCommand
	OleDB / OleDbCom
	Odbc / OdbcCommand
DataAdapter	SqlClient / SqlDataAdapter
	OracleClient / OracleDataAdapter
	OleDB / OleDbDataAdapter
	Odbc / ObdcDataAdapter
DataReader	SqlClient / SqlDataReader
	OracleClient / OracleDataReader
	OleDB / OleDbDataReader
	Odbc / OdbcDataReader

Tabela 1. Lista com o nome das classes dos principais providers ADO.NET

ADO.NET Conectado e Desconectado

Existem dois modelos de se trabalhar a ligação do software com o banco de dados no ADO.NET: O modelo conectado e o modelo desconectado.

No modelo conectado o banco de dados trabalha de forma online, ou seja, em cada operação CRUD realizada será aberta uma conexão com o banco de dados, e esta conexão será fechada após o seu uso, a grande vantagem de utilizar esse modelo é que sempre se tem certeza de que os dados manipulados são atuais, porém a desvantagem é que o constante acesso a rede torna o processo mais lento e o desempenho da aplicação pode cair de forma considerável. Em alguns casos o uso deste modelo é obrigatório, pois os dados manipulados devem estar sempre

atualizados.

No modelo desconectado os dados serão manipulados sem a necessidade de utilizar os serviços de rede a todo o momento. Utilizar o modelo desconectado não quer dizer que jamais será conectado ao banco de dados, mas sim que o sistema buscará todos os necessários em um determinado momento e feito isso ele irá manipular estes dados em memória e então com todas as operações realizadas nos dados em memória o sistema se conecta novamente ao banco de dados e sincroniza as informações (entre o banco de dados e os dados em memória). A desvantagem deste modelo é trabalhar com dados antigos e a vantagem é que desta forma é reduzido o número de acessos ao banco de dados.

Geralmente no modelo desconectado os únicos objetos utilizados para realizar as operações no sistema são o SqlConnection, o DataSet e o SqlDataAdapter. Na **Listagem 5** é possível ver um DataSet sendo populado com o auxílio de um DataAdapter.

Listagem 5. Populando um DataSet através de um DataAdapter

```
01 string connectionString = @"Data Source=.\SQL;Initial
       Catalog=ArtigoAdoNet;Integrated Security=True";
02
03 string query = @"SELECT * FROM Produto";
04
05 SqlConnection conn = new SqlConnection(connectionString);
06
07 DataSet ds = new DataSet();
08
09 SqlDataAdapter adapter = new SqlDataAdapter(query, conn);
10
11 adapter.Fill(ds);
12
13 DataTable dtProduto = ds.Tables[0];
14
15 foreach (DataRow row in dtProduto.Rows)
16 {
```

```
17 Console.WriteLine(row["id"] + " "  + row["descricao"] + " " +
       row["qtde"]);
18 }
```

Um DataAdapter faz o papel de ponte, preenchendo um DataSet com os dados retornados do banco de dados através do método **Fill()**. Um **DataTable** é utilizado para representar uma tabela recebendo o primeiro item da lista e através do atributo Rows o DataTable apresenta todas as linhas da tabela.

Não existe uma forma de avaliar qual destes 2 modelos é melhor, o que deve ser levado em consideração é qual modelo atende melhor a situação imposta pelo sistema.

ADO.NET Entity Framework e LINQ

A ADO.NET Entity Framework abstrai o esquema de um banco de dados relacional e apresenta os dados em um modelo conceitual possibilitando que seja feito um mapeamento objeto relacional (ORM) permitindo o mapeamento de tabelas no banco de dados e as disponibilizando em objetos de nossa aplicação Desta forma desenvolvedores que possuem conhecimento em orientação a objetos podem contar com este mecanismo de acesso a dados sem precisar se aprofundar no SQL. Na **Figura 2** é possível visualizar toda a arquitetura que faz parte do ADO.NET Entity Framework.

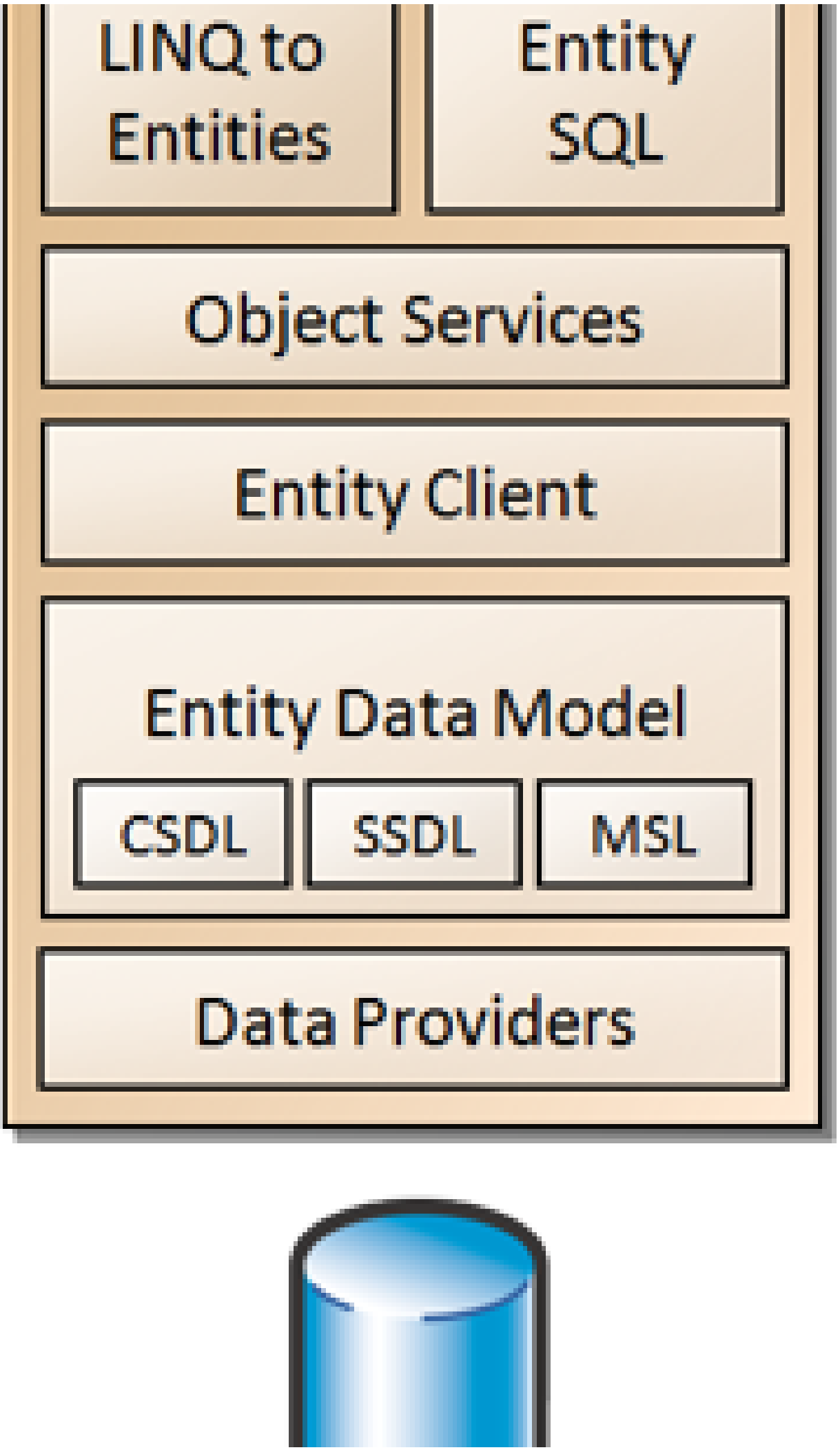

LINQ to Entities
Entity SQL
Object Services
Entity Client
Entity Data Model
CSDL
SSDL
MSL
Data Providers

Figura 2. Arquitetura da tecnologia ADO.NET Entity Framework

Na **Figura 2** o Data Source representa a arquitetura de um banco onde os dados estão armazenados, os **Data Providers** são acessados por um ADO.NET Data Provider, referente ao banco de dados utilizado no Data Source. O **Entity Data Model (EDM)** é um conceito muito importante do ADO.NET Entity Framework e este é constituído por três partes:

O **Conceptual schema definition language (CSDL)** é onde se define e se declara itens como as entidades, associações e herança, sendo que as Entity classes são geradas a partir do mesmo.

O **Store schema definition language (SSDL)** que é onde descreve o container de armazenamento (banco de dados) que persiste os dados.

O **Mapping specification language (MSL)** que é onde são mapeadas as entidades do arquivo CSDL para as tabelas descritas no arquivo SSD.

O **Entity Client** é um provider gerenciado do ADO.NET que suporta acesso aos dados do EDM, o **Object Services** é um componente que permite realizar operações CRUD na base de dados, o **Entity SQL (ESQL)** deriva da T-SQL e é projetada para consultar e manipular entidades do EDM, permitindo tanto que componentes do tipo Entity Services como Entity Client possam executar instruções SQL. Por fim o **LINQ To Entities** é uma linguagem tipada para consultar entidades definidas no EDM.

O **LINQ (Language Integrated Query)** oferece a possibilidade de realizar consultas a partir de objetos. Sendo considerado um elo ligação entre os objetos e os dados no desenvolvimento .NET, o LINQ possui um conceito de LINQ Providers que permite realizar operações básicas de CRUD além de permitir também o mapeamento de tipos definidos pelo usuário, estes providers são: LINQ To SQL, LINQ to DataSet, LINQ to Objects, LINQ to XML e LINQ To Entities. Neste caso será abordado apenas o LINQ To Entities que é o modelo que faz parte do ADO.NET Entity Framework.

Com o conceito de LINQ to Entity Framework é possível realizar o mapeamento de diversas bases de dados, como SQL Server, Oracle, DB2, MySQL, PostgreSQL e SQLite, além de fontes como XML e serviços. O LINQ To Entities nos permite trabalhar em alto nível de abstração de dados do esquema relacional, suportando o modelo EDM para definição de dados no nível conceitual e escrita de consultas usando a linguagem C#. Na **Tabela 2** abaixo é possível ver algumas instruções do LINQ escritas em C#.

Cláusula	Descrição
From	Especifica uma fonte de dados em forma de variável.
Where	Filtra os elementos da fonte de dados baseado em uma condição booleana.
Select	Permite especificar o tipo e o formato dos elementos que serão retornados na consulta.
Join	Realiza a união de duas fontes de dados baseado em comparações entre os elementos.
Equals	Substitui o operador "==" na comparação.
Group	Agrupa os resultados de uma consulta.
Into	Referência para os resultados de uma cláusula Join, Group ou Select.
Orderby	Classifica a ordem dos resultados na direção crescente ou decrescente.
Let	Aplica uma variável que pode ser reutilizada na consulta.

Tabela 2. Instruções do Linq

Na **Listagem 6** é possível visualizar o exemplo de um select simples aplicando um filtro de retorno, utilizando o LINQ.

Listagem 6. Select baseado na tecnologia LINQ

```
var produtos = from produto in lstProdutos
  where produto.Qtde > 10
  orderby produto.Descricao
  select produto;
```

Nessa listagem os produtos que possuem a quantidade maior que 10 em estoque são adicionados a variável produtos.

Um conceito muito utilizado dentro da tecnologia LINQ é o conceito de **Lambda Expressions** que pode ser definida como uma função anônima que contém expressões e sentenças, podendo criar delegates ou árvores de tipos de expressões. Métodos anônimos facilitam pesquisas em coleções genéricas de objetos ou qualquer valor que deseje ser listado, quanto mais complexa é a busca ao banco de dados mais linhas de código nosso programa irá possuir, sendo assim as lambda expressions podem auxiliar neste problema, criando consultas mais simplificadas. A sintaxe das lambda expressions é a seguinte:

<parâmetros de destino> => <expressão lógica>

Os parâmetros de destino podem receber qualquer tipo de dado, o sinal “=>” é o mesmo que “vai para”, a expressão lógica pode ser uma comparação de valores, uma expressão matemática, etc. Na **Listagem 7** a mesma consulta feita na **Listagem 6** está representada utilizando Lambda Expressions.

Listagem 7. Select utilizando Lambda Expressions

```
var produtos = lstProdutos.Where(param => param.Qtde > 10).
OrderBy(param => param.Descricao).ToList();
```

Na **Listagem 7** é atribuído o retorno da consulta a variável produtos, porém dentro da Lambda Expression quem recebe o valor que irá ser o parâmetro para verificação da condição é o “param”, sendo o método ToList() usado para retornar um List<Produto> para a variável produtos.

ADO.NET e SQL Server

O **Microsoft SQL Server** comporta altas cargas de trabalho corporativas, pois possui um alto nível de desempenho, disponibilidade, segurança e é muito utilizado em aplicações da plataforma .NET devido ao fato do próprio Visual Studio possuir inúmeras ferramentas para manipulação e auxilio não só do banco de dados criados em SQL Server mas também de componentes que podem ser utilizados dentro das aplicações. O ADO.NET mesclado com o Microsoft SQL Server oferece um suporte desde aplicações pequenas até aplicações corporativas de grande porte, através do Serviços de Objetos do ADO.NET o suporte fica ainda mais eficiente e otimizado.

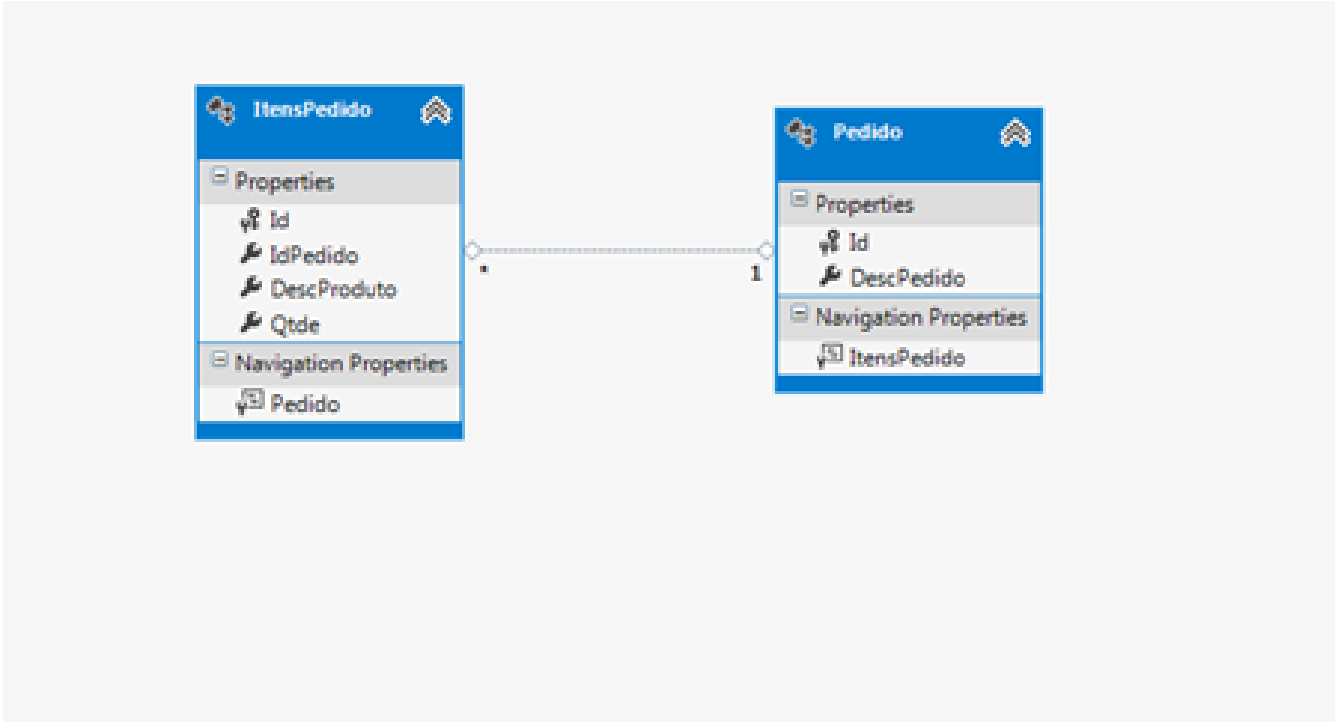
O Provider disponível para acesso ao banco de dados do tipo SQL Server possui o prefixo “Sql” como foi visto na tópico “Providers”, o seu namespace é o System.Data.SqlClient, um exemplo seria uma instância da conexão do ADO.NET com o banco de dados SQL Server, que pode ser manipulada pela classe System.Data.SqlClient.SqlConnection. As demais classes já vistas para realizar operações também se encontram dentro deste namespace System.Data.SqlClient.

Construindo um exemplo prático com ADO.NET

O exemplo prático será a criação de um sistema simples, que mantém Pedidos e

Adiciona Itens a estes pedidos, utilizando as tabelas “Pedido” e “ItensPedido”, a tecnologia utilizada para persistir os dados será baseada na criação de classes DAO, onde a persistência dos dados será implementada com os conceito do **ADO.NET**, utilizando as classes **SqlConnection** e **SqlCommand**.

O primeiro passo é a criação de um banco de dados do SQL Server. Para isso é necessário que o SQL Server esteja devidamente instalado e para confirmar se o serviço do SQL Server está ativo basta verificar pelos “**Services.msc**”, disponibilizados pelo Windows XP/7/8. O Server Explorer também é disponibilizado nas outras versões do Visual Studio. A criação deste banco pode ser feita pelo próprio **Visual Studio 2012**, em uma aba conhecida como **Server Explorer**, este banco de dados será criado localmente no endereço (localhost)\SQLEXPRESS. A estrutura utilizada neste projeto pode ser vista no MER (Modelo Entidade-Relacionamento) da **Figura 3**.



abrir imagem em nova janela

Figura 3. Estrutura de classes do banco de dados

Na **Figura 3** nota-se o relacionamento entre Pedido e ItensPedido, onde um pedido irá

possuir vários itens e cada item fará parte de apenas um pedido.

Criando as Classes do tipo DAO

O próximo passo é a criação de classes baseadas no conceito de DAO, onde todas as operações referentes a banco de dados serão executadas, desta forma o código de tela ficará separado do código de acesso a dados, apenas quem irá tratar interações com o banco de dados é a classe DAO. Na **Listagem 8** temos o código da classe PedidoDAO.

Listagem 8. Classe GenericDAO e suas operações

```
01 public class PedidoDAO
02 {
03        SqlConnection connection = new
            SqlConnection(Program.ConnectionString);
04
05        public PedidoDAO()
06        {
07            connection.Open();
08        }
09
10        public void Add(string descricao)
11        {
12            SqlCommand sql = new SqlCommand("Insert into Pedido
                (DescPeiddo) values(" + descricao + ")", connection);
              sql.ExecuteNonQuery();
13        }
14
15        public List<string> List()
16        {
17            List<string> list = new List<string>();
18
19            SqlCommand sql = new SqlCommand("Select * from Pedido",
                  connection);
20            SqlDataReader reader = sql.ExecuteReader();
21
22            while (reader.Read())
23            {
24                list.Add(reader["DescPedido"].ToString());
```

```
25            }
26
27            return list;
28        }
29
30        public void Update(string id, string descricao)
31        {
32            SqlCommand sql = new SqlCommand("Update Pedido set DescPedido
                = "+ descricao + "Where id = " + id +"", connection);
              sql.ExecuteNonQuery();
33        }
34
35        public void Delete(string id)
36        {
37            SqlCommand sql = new SqlCommand("delete from Pedido where id =
                  " + id + "", connection);
38            sql.ExecuteNonQuery();
39        }
40 }
```

Na linha 03 é criado um objeto do tipo **SqlConnection**, que será responsável pela conexão com o banco de dados, na instância do objeto a string de conexão é fornecida. Nos métodos de ação com o banco de dados é criado um objeto do tipo **SqlCommand**, que é responsável por criar e executar as interações com o banco de dados, este recebe por parâmetro a instrução SQL e o objeto de conexão.

Na linha 12 temos o método para inserção de novos pedidos onde passamos a instrução SQL Insert para o command e invocamos o método ExecuteNonQuery, que também é utilizado nos casos de delete e update, visto que não precisaremos ter retorno de dados.

Na linha 15 temos o método responsável por retornar todos os pedidos cadastrados. Nestes casos percorremos todo o DataReader e adicionamos o resultado em um List.

Agora basta criar os formulários que irão utilizar estas operações CRUD já definidas dentro do nosso DAO, na **Figura 4** é possível ver os dois formulários de CRUD criados

no projeto, chamados de **PedidoCRUD** e **ItensPedidoCRUD**.

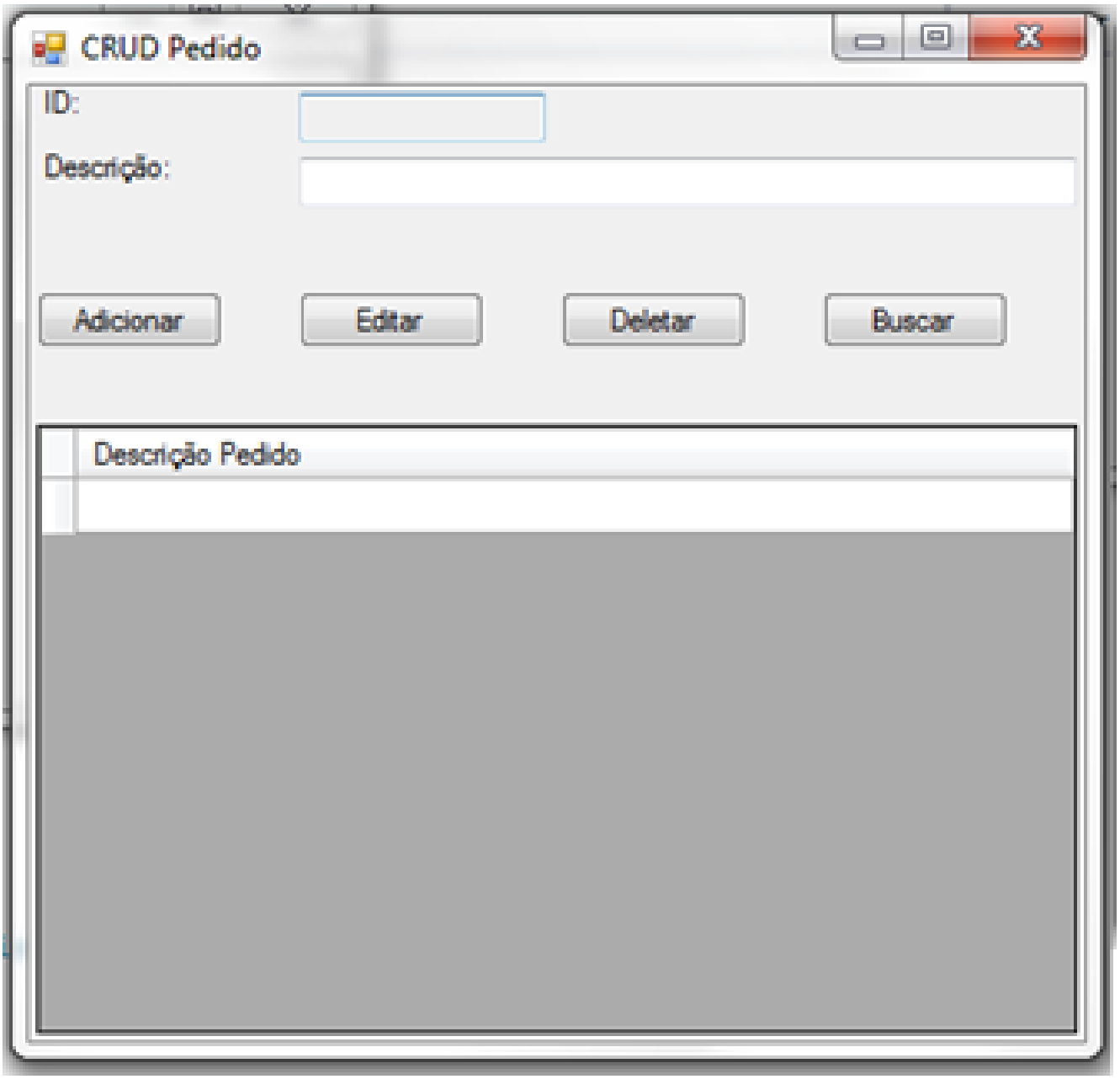


Figura 4. Telas de CRUD de Pedido

Na **Figura 4** nota-se que os componentes utilizados para a criação das telas são relativamente simples, estes seriam **Label**, **TextBox**, **Buttons** e **DataGridView**.

Agora basta instanciar uma classe DAO dentro do formulário e utilizar seus recursos disponíveis para interagir com o banco de dados. Para utilizar os métodos CRUD da DAO precisamos instanciar a mesma como mostrado a seguir:

PedidoDAO pedidoDao = new PedidoDAO();

Na **Listagem 9** é possível visualizar os eventos dos botões “Adicionar” e “Buscar”, do formulário PedidoCRUD.

Listagem 9. Eventos dos botões Adicionar e Buscar

```
private void _btnBuscar_Click(object sender, EventArgs e)
  {
     _dgvPedidos.DataSource = pedidoDao.List();
  }

  private void _btnAdicionar_Click(object sender, EventArgs e)
  {
     string descricao = _txtDescricao.Text;

     pedidoDao.Add(_txtDescricao.Text);
  }
```

No evento do botão buscar é utilizado o método **List()** da classe PedidoDAO e adicionado ao “DataSource” do DataGridView que lista os pedidos. No evento do botão Adicionar um novo registro na tabela Pedido é cadastrado, baseado no que foi inserido no TextBox utilizando o método do PedidoDAO chamado **Add()**.

Conclusão

Neste artigo vimos os principais conceitos sobre a arquitetura do ADO.NET, nos permitindo ter uma base para uso da mesma para diversas fontes de dados. O uso direto da plataforma ADO.NET faz com que tenhamos maior controle do que está sendo executado na base de dados, além de termos melhor performance nas operações isoladas, visto que não temos uma camada de mapeamento, como ocorre nos frameworks ORM.

Cada projeto possui suas características e é fundamental avaliarmos a mesma antes de decidirmos qual estratégia de acesso a dados utilizar. Em alguns casos onde o desempenho é primordial e milissegundos farão grande diferença, é recomendável utilizar ADO.NET. Em outros cenários onde podemos abrir mão destes milissegundos em troca de maior produtividade e de uma manutenção mais fácil, podemos adotar um framework ORM.

Links

Artigo - Using ADO.NET programmatically with C# - Umut ŞİMŞEK

http://www.codeproject.com/Articles/12381/Using-ADO-NET-programmatically-with-C

Artigo – Uma visão geral do ADO.NET - Lucas Húngaro

http://www.linhadecodigo.com.br/artigo/435/uma-visao-geral-do-adonet.aspx

Documento – 10 Razões para adotar LINQ nas aplicações .NET - Renato Haddad

http://msdn.microsoft.com/pt-br/library/dd890987.aspx

Artigo - LINQ (consulta integrada à linguagem) – Wiki MSDN

http://msdn.microsoft.com/pt-br/library/bb397926.aspx

Artigo - Introdução ao ADO.NET Entity Framework – Ramon Durães

http://www.linhadecodigo.com.br/artigo/1834/introducao-ao-adonet-entity-framework.aspx

DevMedia

A DevMedia é um portal para analistas, desenvolvedores de sistemas, gerentes e DBAs com milhares de artigos, dicas, cursos e videoaulas gratuitos e exclusivos para assinantes.

Publicado em 2013

O que você achou deste post?

Gostei (10) (0)

+ Mais conteúdo sobre .net

Todos os comentarios (2)

Postar dúvida / Comentário

Meus comentarios

Renan Roncarati.
Show de Bola este Artigo, deu para nos guiar melhor no entendimento da arquitetura ADO.NET, que utilizamos no dia a dia, mas nem sempre entendemos porque ocorre dequela maneira. Parabéns pelo post.

[há +1 ano] - Responder

Douglas Hideyuki Shirasu
Muito bom, estou começando a mexer com ADO.NET e me ajudou a ter uma visão mais clara da sua arquitetura. Parabéns pelo artigo!

[há +1 ano] - Responder

Mais posts

Artigo

Conhecendo as Classes e Objetos de base do ASP.NET

Artigo

Mapeamento Objeto-Relacional com NHibernate

Revista

Revista Easy .net Magazine Edição 41

Artigo

Vantagens e Desvantagens da POO

Artigo

Windows Services: Desenvolvendo serviços do Windows

Artigo

Deploy de aplicações com o Visual Studio

Listar mais conteúdo