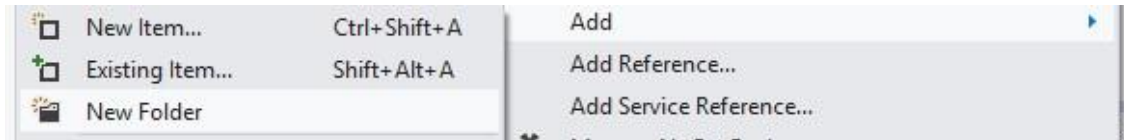


BANCO DE DADOS EM c Sharp USANDO EF

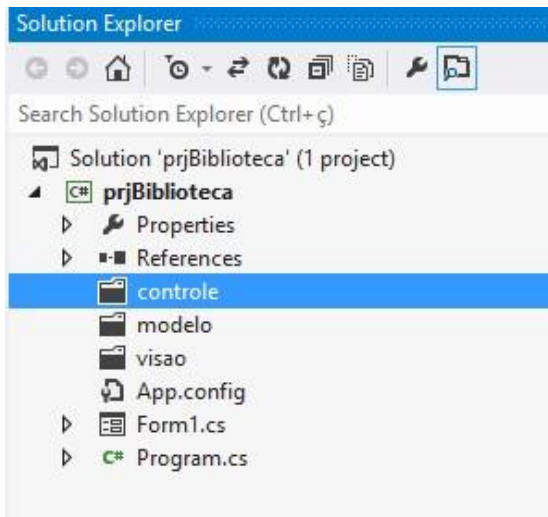
1. Crie um projeto chamado .

Name:	prjBiblioteca
Location:	H:\
Solution:	Create new solution
Solution name:	prjBiblioteca

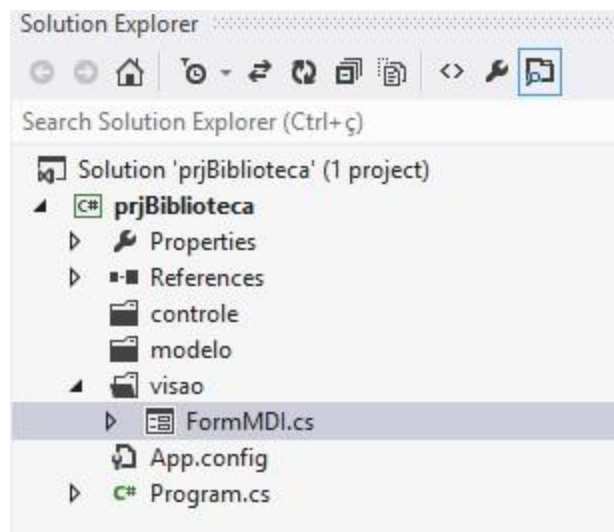
2. Crie três pastas chamadas modelo, controle e visão dentro do projeto:



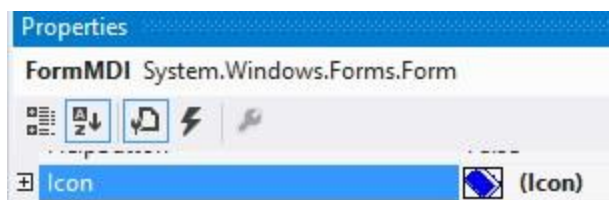
O resultado deve ser:



Exclua o formulário Form1 e crie um novo formulário FormMDI dentro da pasta visão:



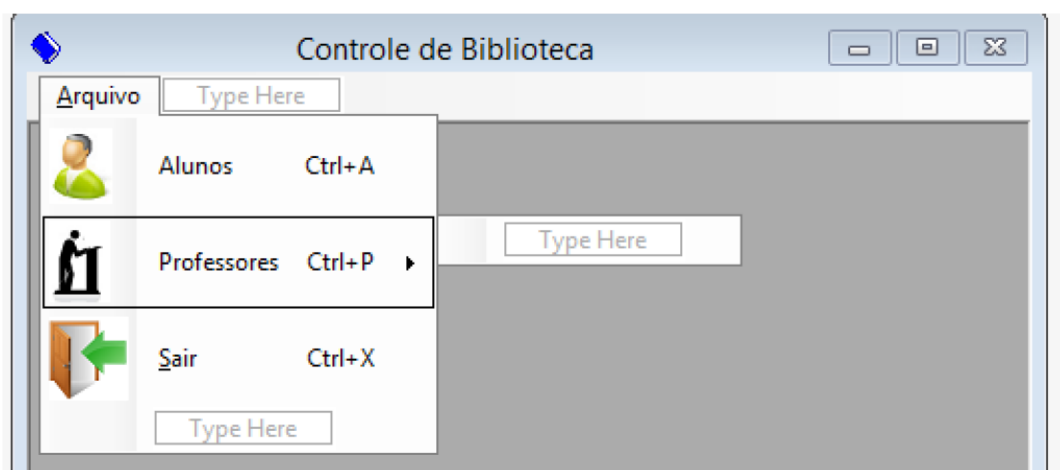
Troque as propriedades do formulário para os seguintes valores:



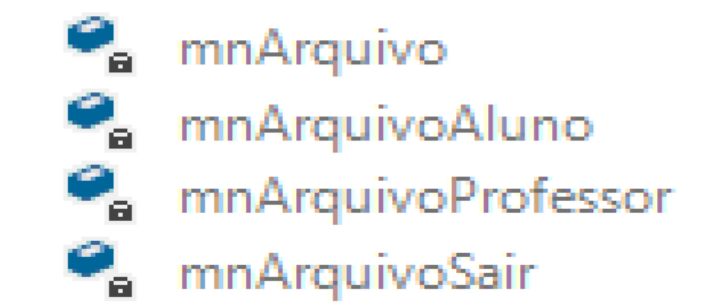
IsMdiContainer	True
----------------	------

StartPosition	WindowsDefaultLocation
WindowState	Maximized
Text	Controle de Biblioteca

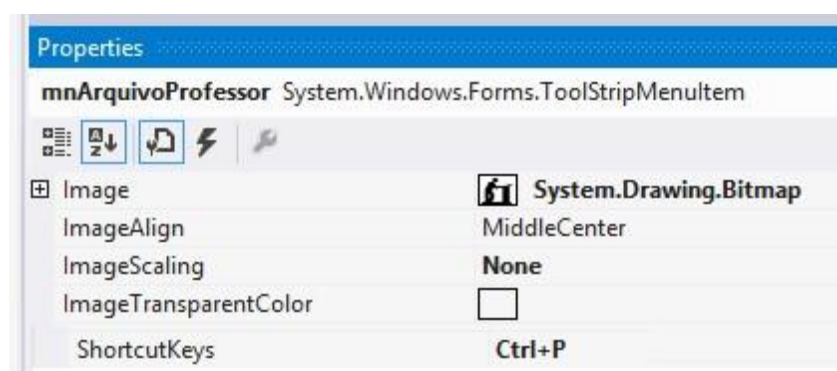
Crie o menu para o sistema:



Siga os seguintes nomes para o :



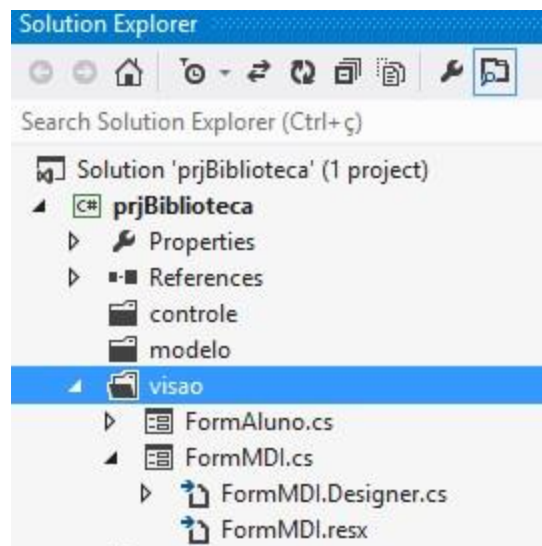
Configure as imagens e defina as teclas de atalho para todos os itens como no exemplo abaixo, onde configuramos para a opção professor:



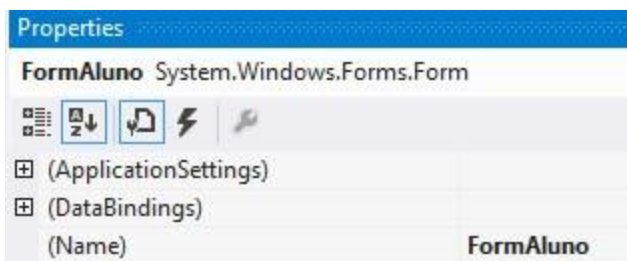
Codifique o evento da opção para o item mnArquivoSair, evento click:

```
private void mnArquivoSair_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Deseja Sair", "Alerta",
        MessageBoxButtons.YesNo, MessageBoxIcon.Question)
        == DialogResult.Yes)
    {
        System.Environment.Exit(0);
    }
}
```

Projetaremos agora o Formulário de Aluno. Adicione um novo formulário em visão chamado **FormAluno**:



Altere as seguintes propriedades:

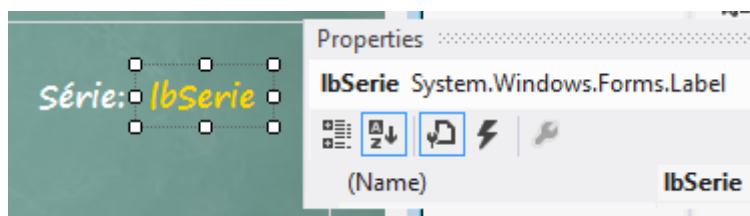
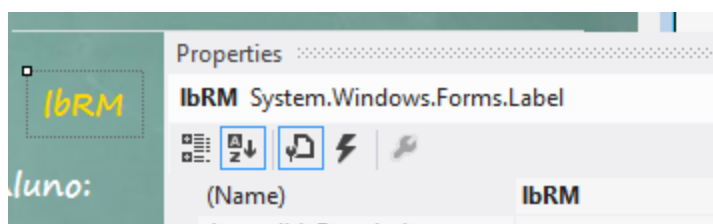


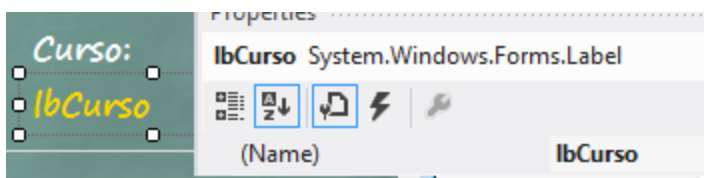
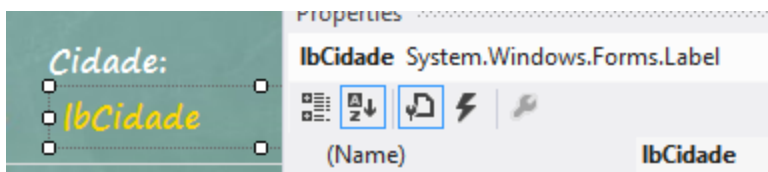
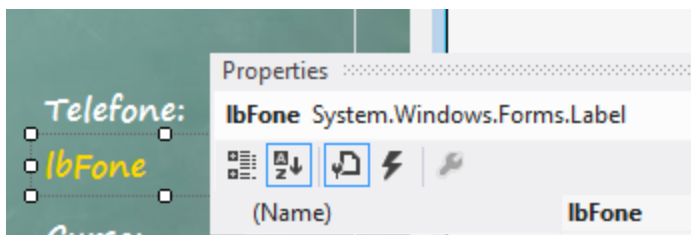
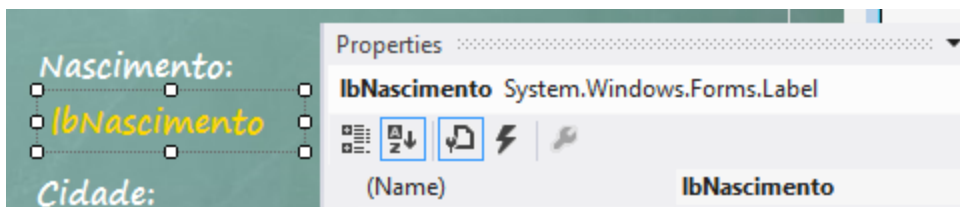
FormBorderStyle	FixedSingle
HelpButton	False
Icon	(Icon)
MaximizeBox	True
MaximumSize	0; 0
MinimizeBox	False

Vamos desenhar agora os componentes visuais a serem controlados pelo sistema. Comece criando os rótulos de exibição dos campos da tabela, no nosso caso, serão o código do aluno, nome, nascimento, sala, conforme a aparência sugerida:

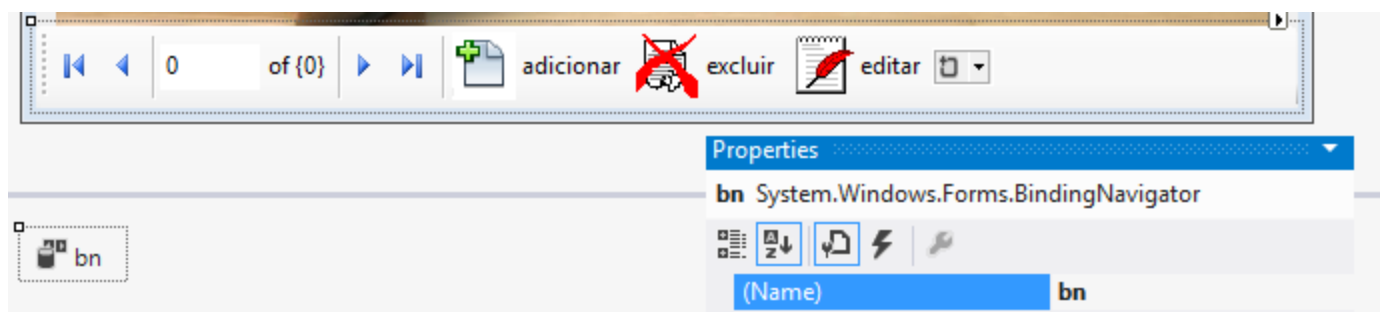


Renomeie os componentes de acordo com os seguintes nomes propostos:

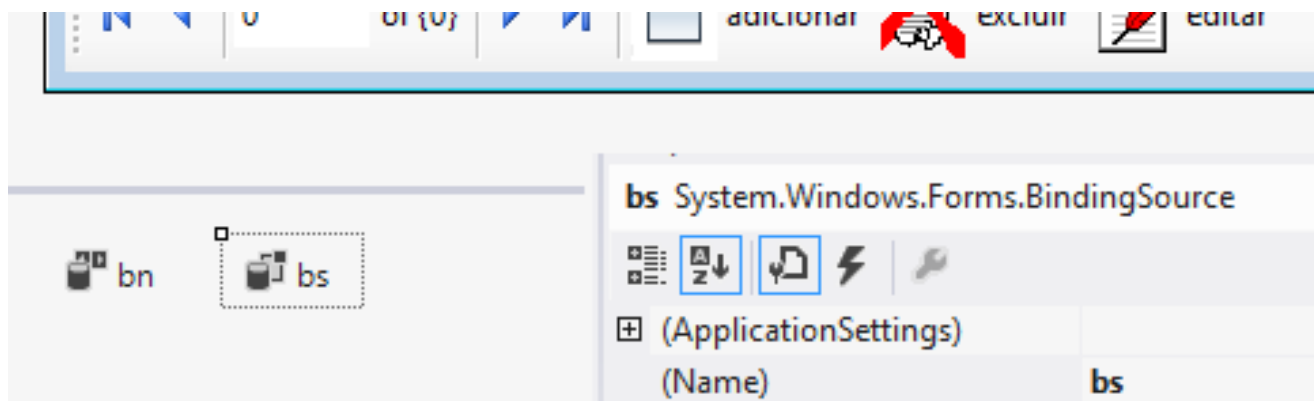




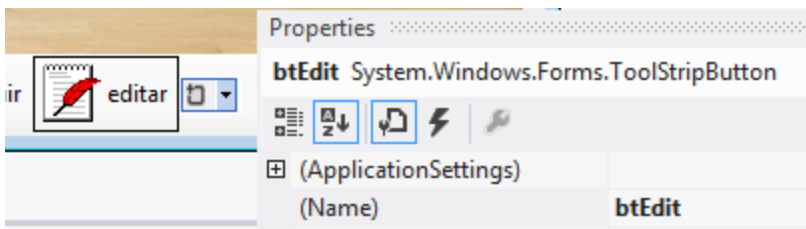
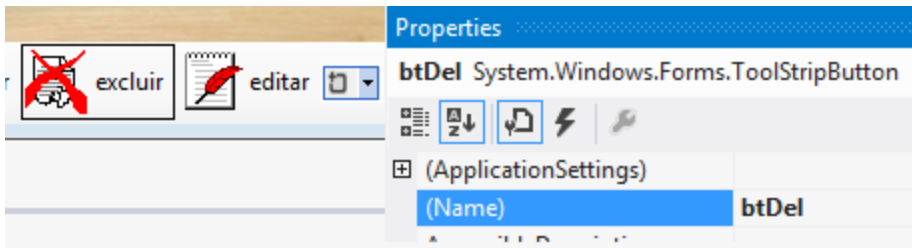
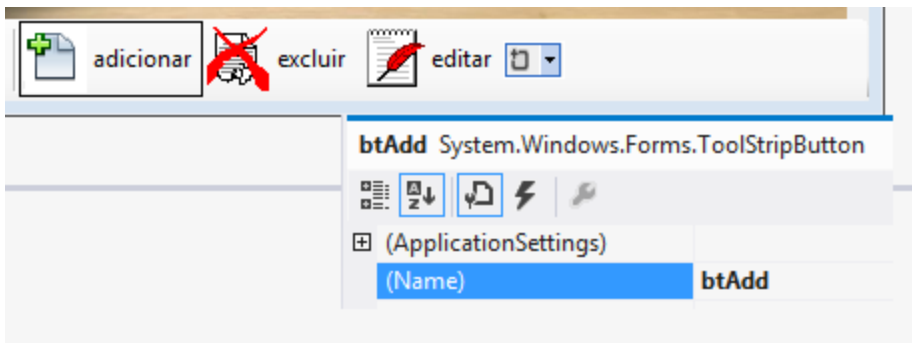
Nome dos componentes de navegação:



Acrescente um componente do tipo **bindingSource**:



Nome dos botões com imagens na barra de navegação:



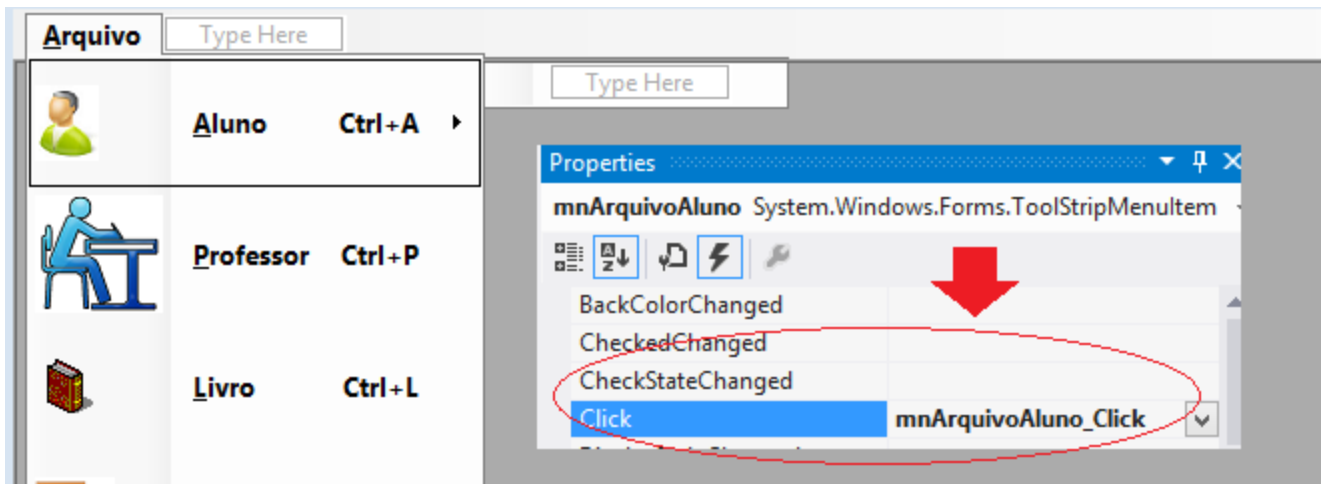
Em **FormMDI** codifique um objeto do tipo **FormAluno** no início da classe com o valor null:

```
namespace prjBiblioteca.visao
{
    public partial class FormMDI : Form
    {
        public visao.FormAluno frAluno = null;

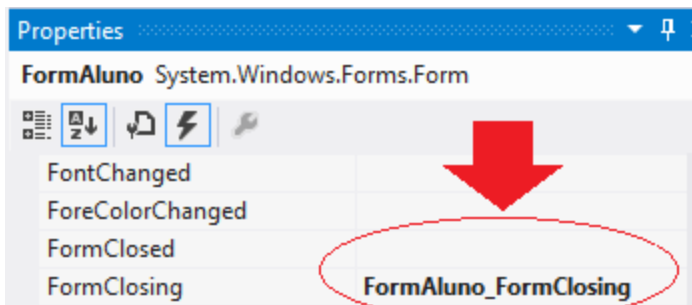
        public FormMDI()
        {
            InitializeComponent();
        }
    }
}
```

A red box highlights the line `public visao.FormAluno frAluno = null;` and a red arrow points to it.

Chame agora o objeto criado no evento do menu que contém a opção de abrir o formulário de alunos:



É necessário agora codificar o evento que controla o fechamento do formulário, removendo-o da memória, evitando que o formulário abra diversas cópias de si mesmo. Em FormAluno, codifique o seguinte método:



```
private void FormAluno_FormClosing(object sender, FormClosingEventArgs e)
{
    // volta em fr o formulário pai usando MdiParent
    FormMDI fr = (FormMDI)this.MdiParent;
    // fr passa a valer null, removendo o mesmo da memória
    fr.frAluno = null;
}
```

Passaremos agora a integração com o EF (Entity Framework). Antes de tudo precisamos de um banco de dados biblioteca com as seguintes tabelas:

```
mysql> desc aluno;
```

Field	Type	Null	Key	Default	Extra
idaluno	int(11)	NO	PRI	NULL	
nome	varchar(45)	NO		NULL	
nascimento	date	NO		NULL	
idcurso	int(11)	NO	MUL	NULL	
endereco	varchar(50)	NO		NULL	
numero	int(11)	NO		NULL	
bairro	varchar(30)	YES		NULL	
uf	varchar(2)	YES		NULL	
cep	varchar(8)	YES		NULL	
cidade	varchar(8)	YES		NULL	
fone	varchar(12)	YES		NULL	
email	varchar(255)	YES		NULL	
sala	varchar(2)	YES		NULL	

```
mysql> desc curso;
```

Field	Type	Null	Key	Default	Extra
idcurso	int(11)	NO	PRI	NULL	
descricao	varchar(45)	YES		NULL	

```
mysql> desc autor;
```

Field	Type	Null	Key	Default	Extra
idAutor	int(11)	NO	PRI	NULL	
nome	varchar(45)	NO		NULL	
nacionalidade	varchar(30)	NO		NULL	
nascimento	datetime	NO		NULL	
ocupacao	varchar(3)	NO		NULL	
telefone	varchar(24)	NO		NULL	

```
mysql> desc editora;
```

Field	Type	Null	Key	Default	Extra
idEditora	int(11)	NO	PRI	NULL	
descricao	varchar(45)	NO		NULL	
estado	varchar(2)	NO		NULL	
telefone	varchar(24)	NO		NULL	
email	varchar(45)	NO		NULL	

```
mysql> desc genero;
```

Field	Type	Null	Key	Default	Extra
idGenero	int(11)	NO	PRI	NULL	
descricao	varchar(45)	NO		NULL	

```
mysql> desc livro;
```

Field	Type	Null	Key	Default	Extra
idLivro	int(11)	NO	PRI	NULL	
titulo	varchar(30)	NO		NULL	
idAutor	int(5)	NO	MUL	NULL	
idEditora	int(5)	NO	MUL	NULL	
nropaginas	int(11)	NO		NULL	
edicao	int(11)	NO		NULL	
publicacao	date	NO		NULL	
resumo	varchar(45)	NO		NULL	
codisbn	varchar(32)	NO		NULL	
idGenero	int(3)	NO	MUL	NULL	

```
10 rows in set (0.02 sec)
```

```
mysql> desc professor;
```

Field	Type	Null	Key	Default	Extra
idprofessor	int(11)	NO	PRI	NULL	
nome	varchar(45)	NO		NULL	
idcurso	int(11)	NO	MUL	NULL	
fone	varchar(12)	YES		NULL	
email	varchar(255)	YES		NULL	

```
mysql> desc tend_bairro;
```

Field	Type	Null	Key	Default	Extra
id_bairro	int(11)	NO	PRI	NULL	
bairro	varchar(50)	NO		NULL	
id_cidade	int(11)	NO	MUL	NULL	

```
3 rows in set (0.02 sec)
```



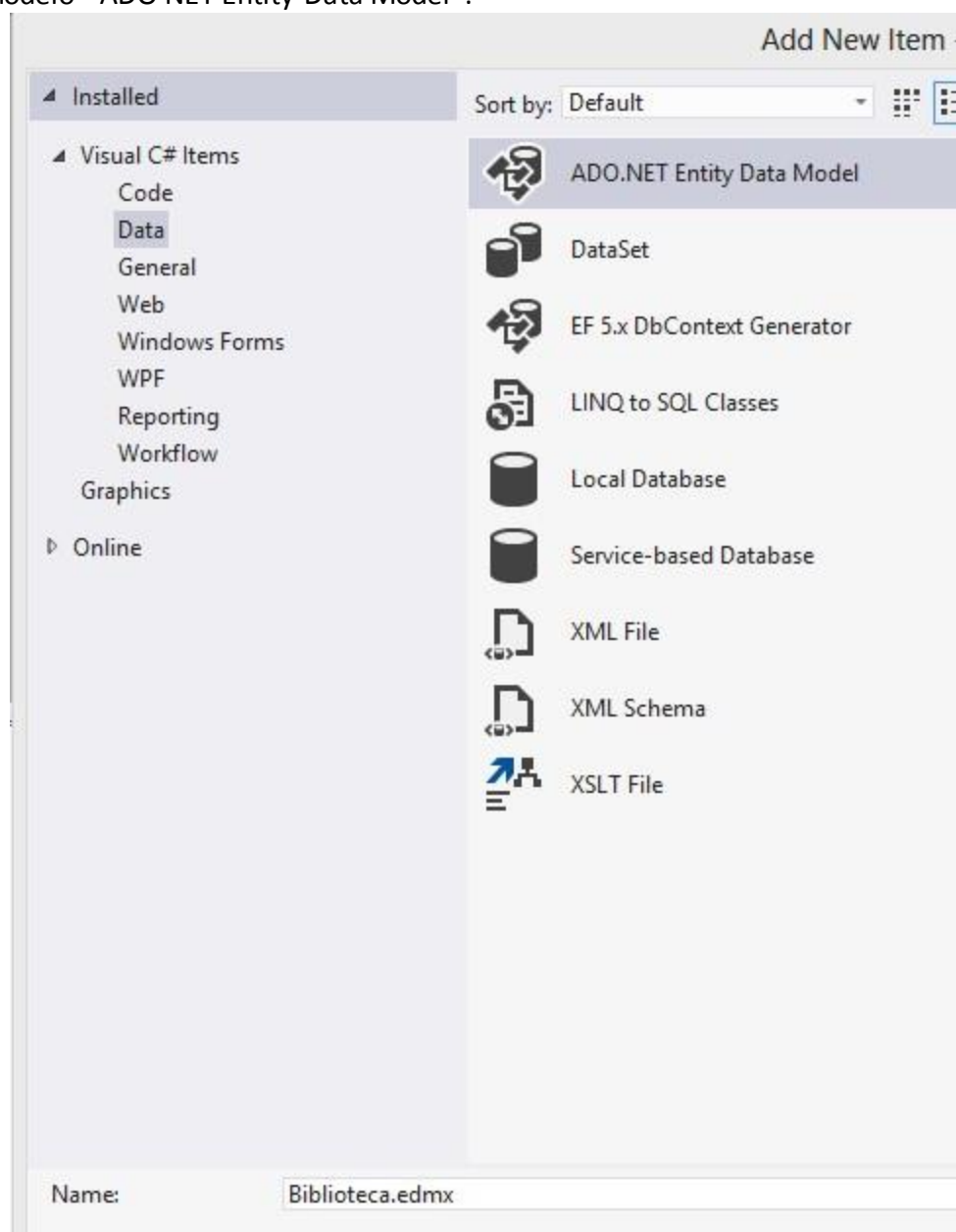
```
mysql> desc tend_cidade;
```

Field	Type	Null	Key	Default	Extra
id_cidade	int(11)	NO	PRI	NULL	
cidade	varchar(100)	NO		NULL	
uf	varchar(2)	NO		NULL	

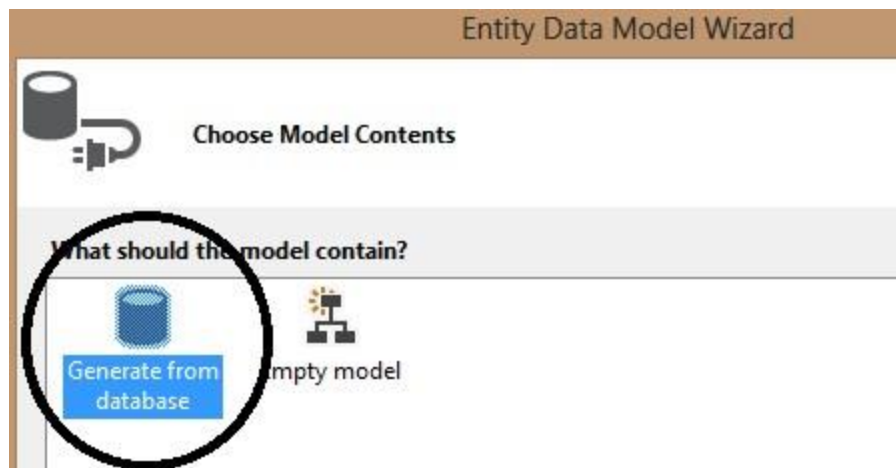
```
mysql> desc tend_endereco;
```

Field	Type	Null	Key	Default	Extra
cep	varchar(10)	NO	PRI		
endereco	varchar(200)	NO	MUL		
id_cidade	int(11)	NO	MUL	NULL	
id_bairro	int(11)	YES	MUL	NULL	

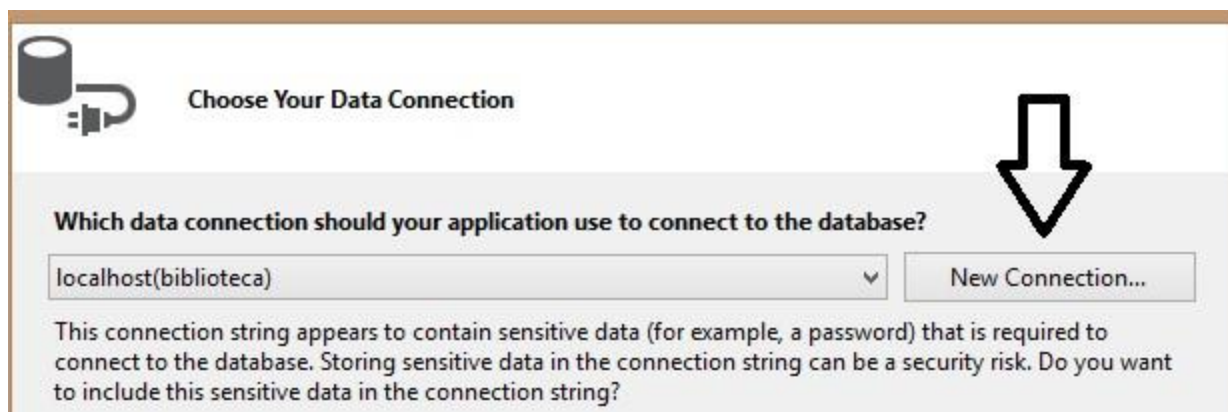
Ou importe o arquivo com extensão SQL que contém o backup do projeto do banco de dados oferecido pelo professor, com tabela de CEP completa (89 MB). Vamos integrar agora um arquivo de MODELO (EDMX) ao projeto. Para isso adicione um novo componente a pasta modelo e de o nome de biblioteca, escolhendo o modelo “ADO NET Entity Data Model”:



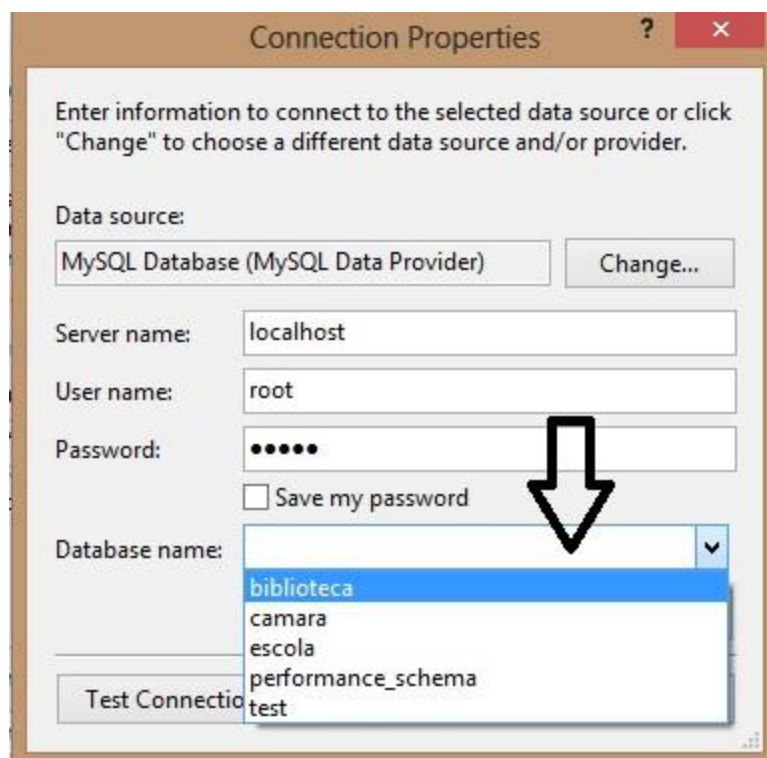
É possível gerar um modelo vazio ou a partir de uma tabela de um banco de dados. Em nosso caso vamos selecionar a partir do nosso banco, então iremos escolher a primeira opção:



Crie uma nova conexão com o banco de dados biblioteca no MYSQL:



Passe os dados para a conexão, como nome do servidor, usuário, senha e o escolha na lista o banco de dados correto e confirme pressionando o botão OK:



Antes de prosseguir, modifique o nome da conexão da Entidade que será registrado em APP.CONF. Caso seja necessário gerar o processo novamente será necessário apagar o arquivo APP.CONF para que o sistema não gere erros: Mande excluir também a string conexão, selecione Não parte superior da caixa de diálogo:

Which data connection should your application use to connect to the database?

localhost(biblioteca) ▼

This connection string appears to contain sensitive data (for example, a password) that can be used to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

☒ No, exclude sensitive data from the connection string. I will set it in my application code.

☐ Yes, include the sensitive data in the connection string.

Entity connection string:

```
metadata=res://*/modelo.Biblioteca.csdl|res://*/modelo.Biblioteca.ssdl|res://*/modelo.Biblioteca.msl;provider=MySql.Data.MySqlClient;provider connection string="server=localhost;user id=root;database=biblioteca"
```

☒ Save entity connection settings in App.Config as:

bibliotecaEntidades

Selecione o banco de dados biblioteca e mude o nome para **bibliotecaModelo**:

Entity Data Model Wizard

Choose Your Database Objects and Settings

Which database objects do you want to include in your model?

☒ Tables

☒ biblioteca

☐ Views

☐ Stored Procedures and Functions

☐ Pluralize or singularize generated object names

☒ Include foreign key columns in the model

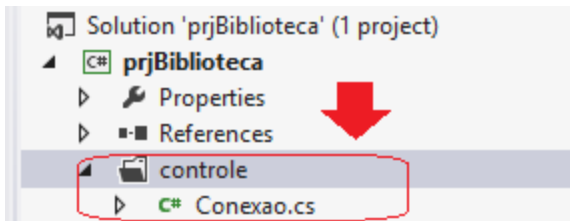
☐ Import selected stored procedures and functions into the entity model

Model Namespace:

bibliotecaModelo

CLASSE DE CONEXÃO COM O BANCO DE DADOS

A forma mais eficiente de controlar a conexão com o banco de dados é criar uma classe que será responsável por esta tarefa. Ela deve ser criada no pacote controle e sua função é passar a string de conexão para o banco de dados, além da senha e usuário de conexão. Criaremos uma classe de conexão em controle:



Codifique as seguintes propriedades e o construtor:

```
class Conexao
{
    private string path;
    private string database;
    private string login;
    private string senha;

    public Conexao(string path, string database, string login, string senha)
    {
        this.path = path;
        this.database = database;
        this.login = login;
        this.senha = senha;
    }
}
```

Codificaremos um método para abrir o banco de dados, passando os dados da string de conexão para o Entity Framework logo abaixo do construtor:

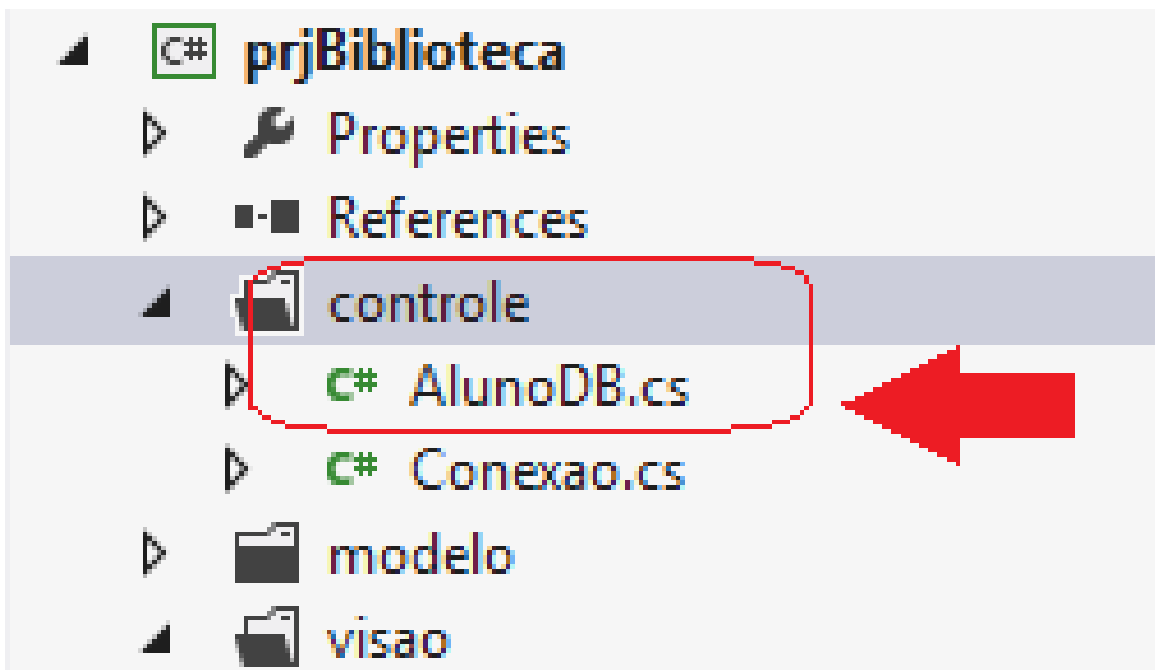
```
public Conexao(string path, string database, string login, string senha)
{
    this.path = path;
    this.database = database;
    this.login = login;
    this.senha = senha;
}
```

```
public String open()
{
    string strcon = "server=" + path + ";"
        + "user id=" + login + ";"
        + "database=" + database + ";" +
        "password=" + senha + ";" +
        "persist security info=True";

    return strcon;
}
```



Codificaremos agora a rotina de consulta ao banco de dados. Será necessário criar uma classe chamada **AlunoDB**, que será responsável pelas ações a serem realizadas no banco de dados:



Agora iremos programar o código de consulta a base de dados e retornar o resultado em um bindingSource que será usado para alimentar a barra de navegação do formulário. Acrescente o método consultar na classe AlunoDB:

```

class AlunoDB
{
    // objeto de conexão a base de dados

    Conexao con = new Conexao("localhost", "biblioteca",
        "root", "minas");

    public void consultar(System.Windows.Forms.BindingSource bs)
    {
        // simplifica a chamada a biblioteca entidades
        using (var banco = new modelo.bibliotecaEntidades())
        {
            // conecta no banco de dados
            banco.Database.Connection.ConnectionString = con.open();
            // consulta usando a linguagem LINQ
            var query = from linha in banco.aluno.Include("curso")
                        orderby linha.idaluno
                        select linha;

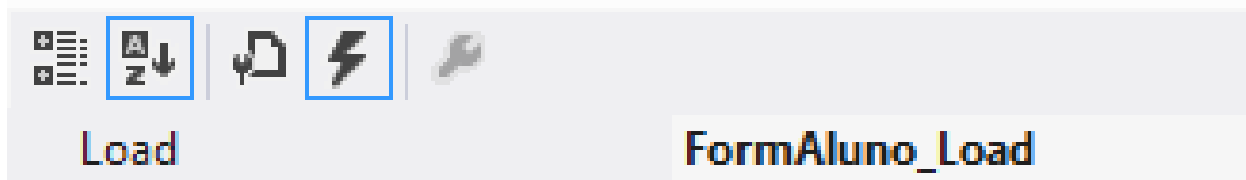
            // transforma em uma lista
            bs.DataSource = query.ToList();
        }
    }
}

```

Perceba o uso da palavra INCLUDE("CURSO"). Ele é usada para garantir que a tabela curso seja consultada e persistida para a lista. Se ela não for usada ao tentar acessar a descrição do curso uma mensagem de erro será exibida dizendo que o objeto foi descartado antes de ser usado pelo bindingSource.

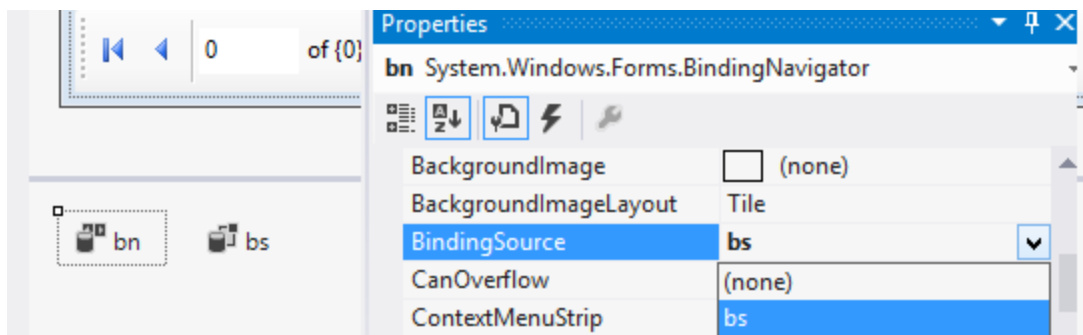
Para testar a conexão vamos realizar a uma consulta na base de dados da tabela aluno. Para isso codificaremos o método **load** de FormAluno conforme abaixo, usando o recurso dos databindings. Cada label será vinculado ao seu respectivo campo na tabela aluno usando a propriedade databindings.

FormAluno System.Windows.Forms.Form



```
private void FormAluno_Load(object sender, EventArgs e)
{
    // conecta ao banco de dados
    controle.AlunoDB aDb = new controle.AlunoDB();
    // retorna a consulta em bs
    aDb.consultar(bs);
    // vincula os labels aos campos das tabelas:
    lbRM.DataBindings.Add(new Binding("text", bs, "idaluno"));
    lbAluno.DataBindings.Add(new Binding("text", bs, "nome"));
    lbFone.DataBindings.Add(new Binding("text", bs, "fone"));
    lbSerie.DataBindings.Add(new Binding("text", bs, "sala"));
    lbCidade.DataBindings.Add(new Binding("text", bs, "cidade"));
    lbNascimento.DataBindings.Add(new Binding("text", bs, "nascimento"));
    lbCurso.DataBindings.Add(new Binding("text",
        bs, "curso.descricao"));
}
```

Para encerrar vincule a barra de navegação com o BindingSource:



Verifique se a tabela de aluno contém algum dado para que não ocorra erro na pesquisa:

```
mysql> select idaluno,nome,sala from aluno;
+-----+-----+-----+
| idaluno | nome       | sala |
+-----+-----+-----+
| 1       | JOAO DOS SANTOS | 2J   |
+-----+-----+-----+
1 row in set (0.00 sec)
```

Verifique se a tabela de curso contém algum dado para que não ocorra erro na pesquisa:

```
mysql> select * from curso;
+-----+-----+
| idcurso | descricao |
+-----+-----+
| 1       | informatica |
+-----+-----+
1 row in set (0.00 sec)
```

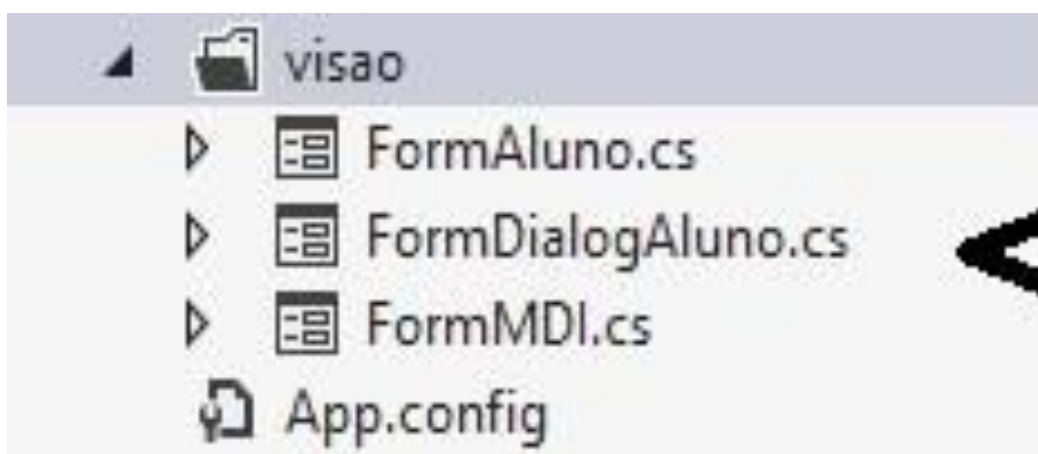
Execute o programa e veja se o aluno é exibido no formulário aluno:



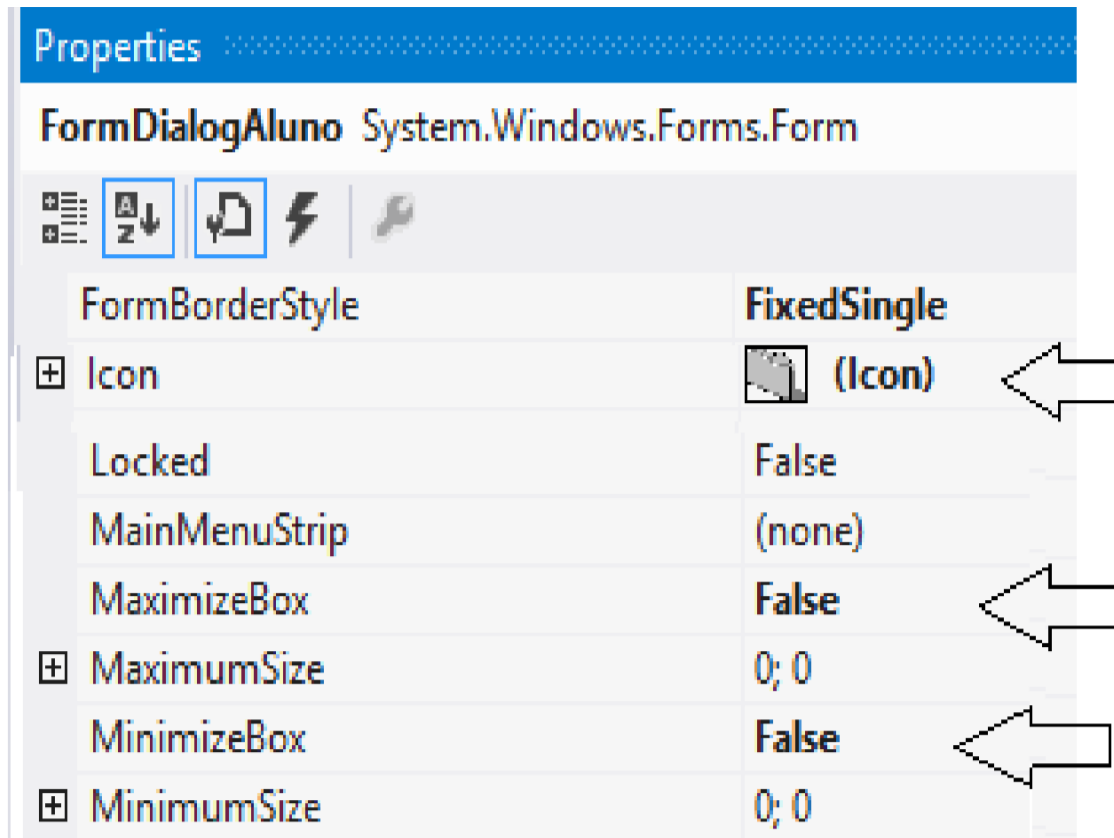
Este código pode não funcionar se a tabela aluno não possuir uma chave estrangeira que relacione um registro com a tabela de cursos, logo se não tem nenhum curso cadastrado na tabela, também não será possível cadastrar o aluno, para isso entre no MYSQL e cadastre um curso para que a rotina seja corretamente executada. Caso nenhum aluno esteja cadastrado cadastre um manualmente na tabela para realizar o teste. Agora deveremos na próxima etapa desenvolver um formulário de cadastro de dados para nosso sistema para realizar a inclusão de registros em nosso banco de dados.

INCLUSÃO DE REGISTROS NO BANCO DE DADOS

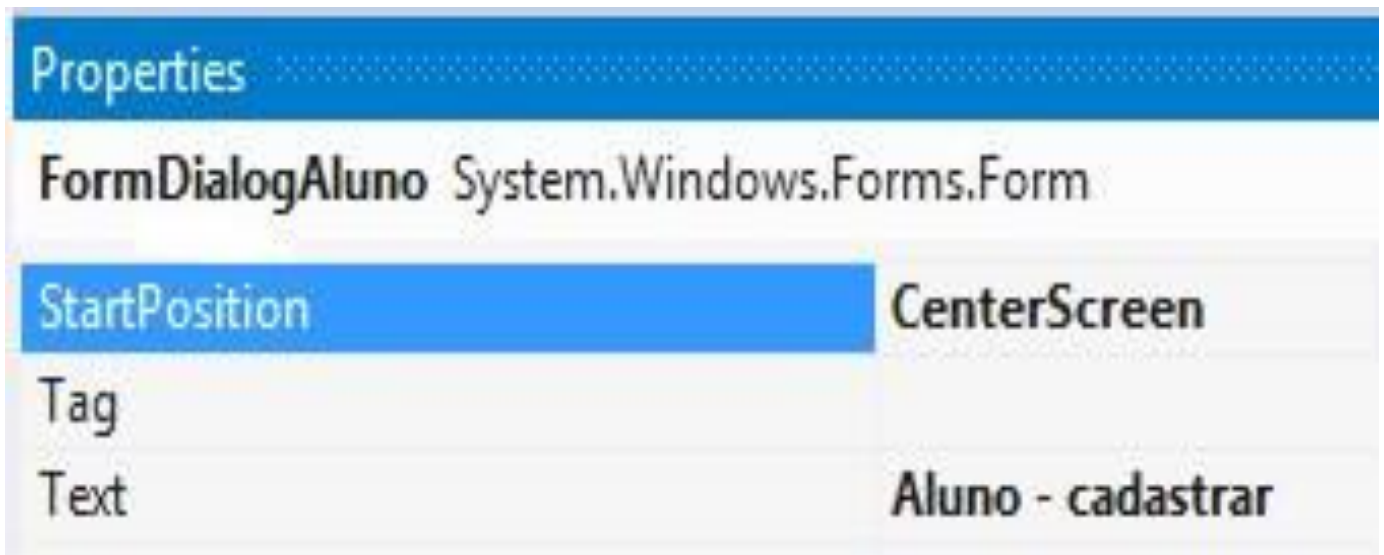
Crie um novo formulário chamado **FormDialogAluno**:



Sete as seguintes propriedades do formulário:



Ajuste também a propriedade para o centro da tela e o texto do formulário:



Desenhe o formulário a seguir:

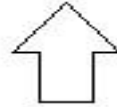
Nome dos campos:

```
.DateTimePicker dtNascimento
.Button btCancelar
.Button btGravar
.GroupBox groupBox1
.MaskedTextBox txtFone
.MaskedTextBox txtCEP
.ComboBox cbUF
.Label label11
.Label label7
.MaskedTextBox txtNumero
.Label label5
.Label label9
.Label label10
.Label label8
.TextBox txtEmail
.Label label6
.TextBox txtCidade
.Label label4
.TextBox txtBairro
.TextBox txtEndereco
.ComboBox cbCurso
.TextBox txtSala
.Label label2
.TextBox txtNome
.Label label12
.Label label3
.Label label1
```

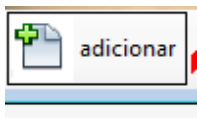
Agora vamos chamar o formulário. Para isso temos que passar o bindingSource para o banco de dados no formulário. Crie a seguinte propriedade em FormDialogAluno:

```
namespace prjBiblioteca.visao
{
    public partial class FormDialogAluno : Form
    {
        private BindingSource bs;

        public BindingSource Bs // recebe o banco de dados
        {
            get { return bs; }
            set { bs = value; }
        }
    }
}
```



No FormAluno, programar o código a seguir:



```
private void btAdd_Click(object sender, EventArgs e)
{
    FormDialogAluno fr = new FormDialogAluno();
    // passa o bindingSource como parametro para o outro form
    fr.Bs = bs;
    // exibe o formulário como caixa de dialogo
    fr.ShowDialog();
}
```

Vamos criar o modelo aluno para que o mesmo seja usado entre os dois formulários. Se aluno contiver o valor nulo, então estamos adicionando dados no banco de dados. Se contiver algum valor não nulo, então estamos editando o campo. Crie a seguinte propriedade aluno (getter e setter) em **FormDialogAluno**:

```
public partial class FormDialogAluno : Form
{
    private BindingSource bs;

    public BindingSource Bs
    {
        get { return bs; }
        set { bs = value; }
    }
}
```

```
private modelo.aluno aluno;

public modelo.aluno Aluno
{
    get { return aluno; }
    set { aluno = value; }
}
```



No botão “adicionar” do FormAluno acrescentar a seguinte linha ao código:

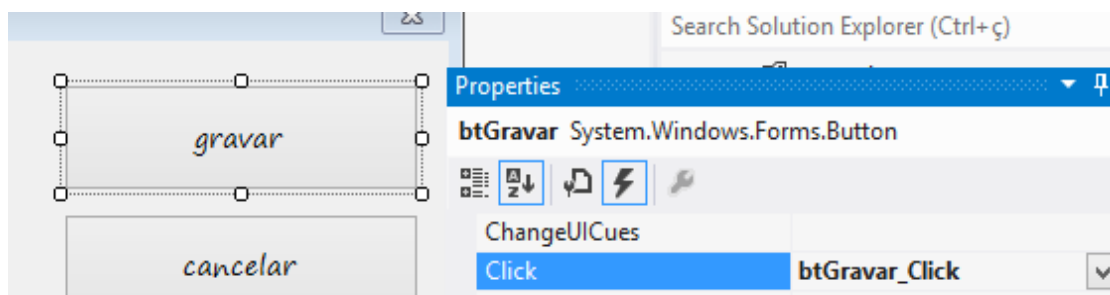
```
private void btAdd_Click(object sender, EventArgs e)
{
    FormDialogAluno fr = new FormDialogAluno();
    // passa o bindingSource como parametro para o outro form
    fr.Bs = bs;

    // inicia aluno com o valor null
    fr.Aluno = null;

    // exibe o formulário como caixa de dialogo
    fr.ShowDialog();
}
```



Volte ao formulário FormDialogAluno e codifique o botão gravar:



```
private void btGravar_Click(object sender, EventArgs e)
{
    if (Aluno == null) {
        novo();
    }
    else
    {
        editar();
    }
}
```

Gere o método “novo” e “editar”:

```
private void editar()
{
    throw new NotImplementedException();
}

private void novo()
{
    // código aqui para adicionar no banco de dados
}
```

Codificaremos o método novo, jogando os campos do formulário para a estrutura de dados e adicionando no banco de dados, logo após limparemos os campos e vamos mover o registro do banco de dados para o último já que adicionamos sempre no final da tabela de dados.

```
private void novo()
{
    // conecta ao banco de dados
    controle.AlunoDB aDB = new controle.AlunoDB();

    // preenche os campos do registro

    Aluno = new modelo.aluno()
    {
        idaluno = aDB.proximoCodigo(),
        nome = txtNome.Text,
        sala = txtSala.Text,
        idcurso = Int16.Parse(
            cbCurso.SelectedValue.ToString()),
        endereco = txtEndereco.Text,
        numero = Int16.Parse(txtNumero.Text),
        cep = txtCEP.Text,
        cidade = txtCidade.Text,
        bairro = txtBairro.Text,
        fone = txtFone.Text,
        uf = cbUF.Text,
        email = txtEmail.Text,
        nascimento = dtNascimento.Value
    };
    // inserir o registro
    aDB.inserir(Aluno);
    // reconstroi a tabela de consulta
    aDB.consultar(Bs);
    // avisa ao bindinsource que a lista mudou
    bs.ResetBindings(false);
    // move para o ultimo registro da tabela
    bs.MoveLast();
}
```

Como não está sendo usada numeração automática, devemos codificar um método chamado próximo código que deve encontrar o maior código, somar um e em seguida atualizar o campo.

Usando a função **MAX ()** na linguagem LINQ é possível realizar a consulta e retornar o valor calculado, convertido em inteiro. Se o banco não tiver nenhum registro o valor retornado será o valor um.

Codifique na classe AlunoDB o seguinte código para o método próximo código:

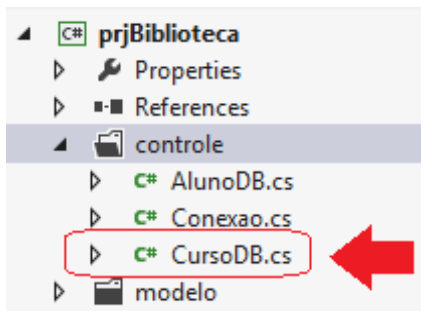
```
public int proximoCodigo()
{
    int t = 0;
    try
    {
        using (var banco = new modelo.bibliotecaEntidades())
        {
            banco.Database.Connection.ConnectionString = con.open();
            var query = (from linha in banco.aluno
                        select linha.idaluno).Max();
            t = Convert.ToInt16(query.ToString());
        }

        return t + 1;
    }
    catch (Exception err)
    {
        System.Console.WriteLine(err.Message);
        return 1;
    }
}
```

Agora também na classe AlunoDB acrescente o método inserir, que gravará o novo registro no banco de dados.

```
public void inserir(modelo.aluno novo)
{
    try
    {
        using (var banco = new modelo.bibliotecaEntidades())
        {
            // conecta ao banco
            banco.Database.Connection.ConnectionString = con.open();
            // adiciona o novo aluno a lista
            banco.aluno.Add(novo);
            // salva no banco de dados
            banco.SaveChanges();
        }
    }
    catch (Exception err) {
        // em caso de erro exibir a mensagem
        System.Windows.Forms.MessageBox.Show("Erro:" + err.Message);
    }
}
```

Agora precisamos resolver o combobox que exibirá do curso, para isso precisamos montar um método que irá exibir a lista de cursos existentes. Crie a classe CursoDB na pasta controle:



Codificar o seguinte método:

```
class CursoDB
{
    Conexao con = new Conexao("localhost", "biblioteca",
        "root", "minas");

    public void listar(System.Windows.Forms.ComboBox cb)
    {
        using (var banco = new modelo.bibliotecaEntidades())
        {
            // conecta ao banco
            banco.Database.Connection.ConnectionString =
                con.open();

            // consulta a lista de cursos cadastrados
            var query = from linha in banco.curso
                        orderby linha.descricao
                        select linha;
            cb.DataSource = query.ToList();
            // exibe a descrição do curso no combobox
            cb.DisplayMember = "descricao";
            // porém ao selecionar retorna o código
            cb.ValueMember = "idCurso";
        }
    }
}
```

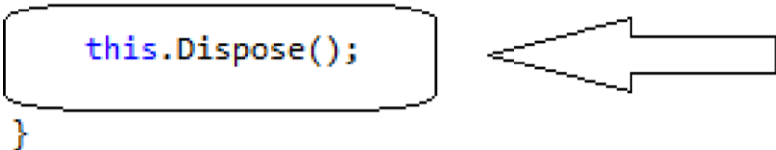
O método acima irá preencher o combobox com a lista de cursos da tabela selecionada. Crie o evento **Load** em **FormDialogAluno** para chamar essa rotina:

```
private void FormDialogAluno_Load(object sender, EventArgs e)
{
    // conecta a tabela curso
    controle.CursoDB cDB = new controle.CursoDB();
    // exibe os cursos no combobox
    cDB.listar(cbCurso);
}
```

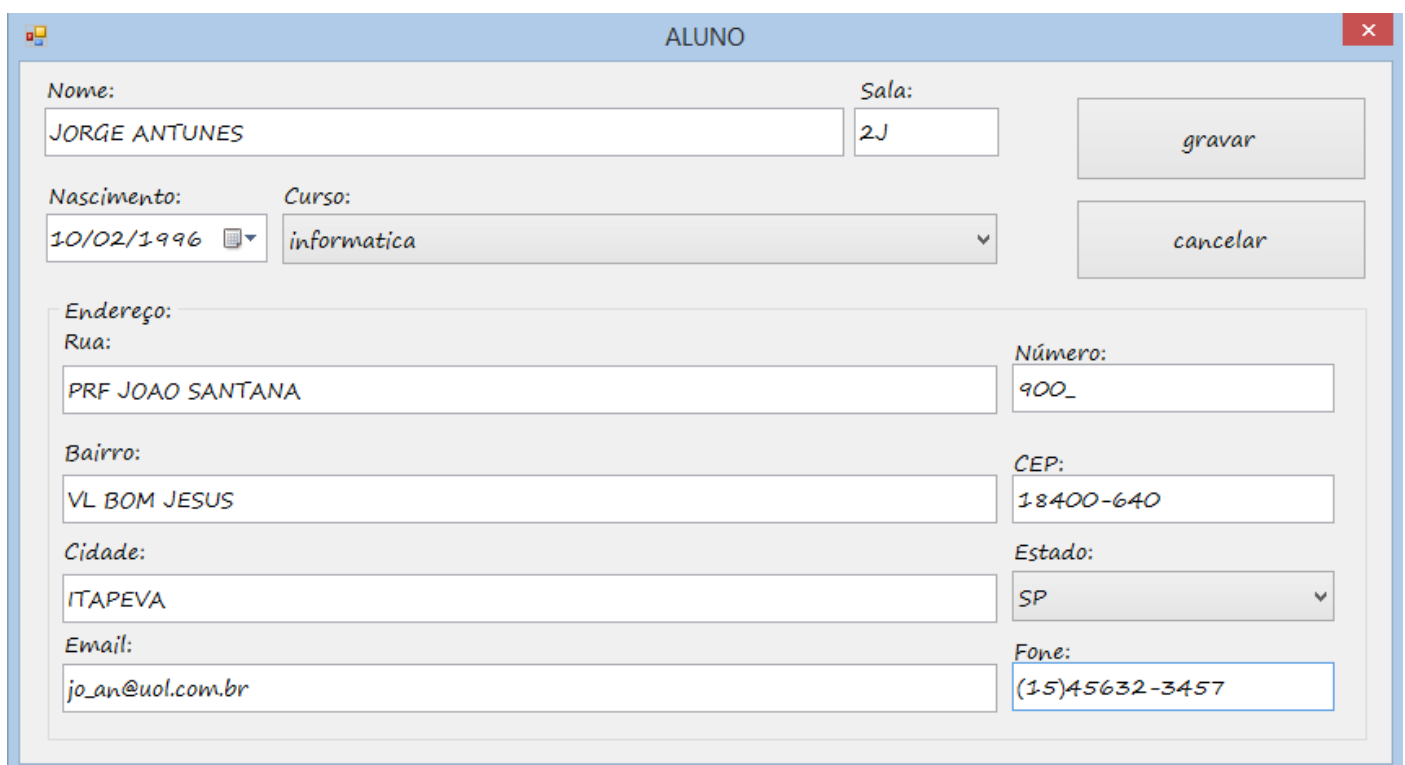
Para finalizar, acrescente a rotina do botão “gravar” o código para fechar o formulário:

```
private void btGravar_Click(object sender, EventArgs e)
{
    if (Aluno == null) {
        novo();
    }
    else
    {
        editar();
    }

    this.Dispose();
}
```

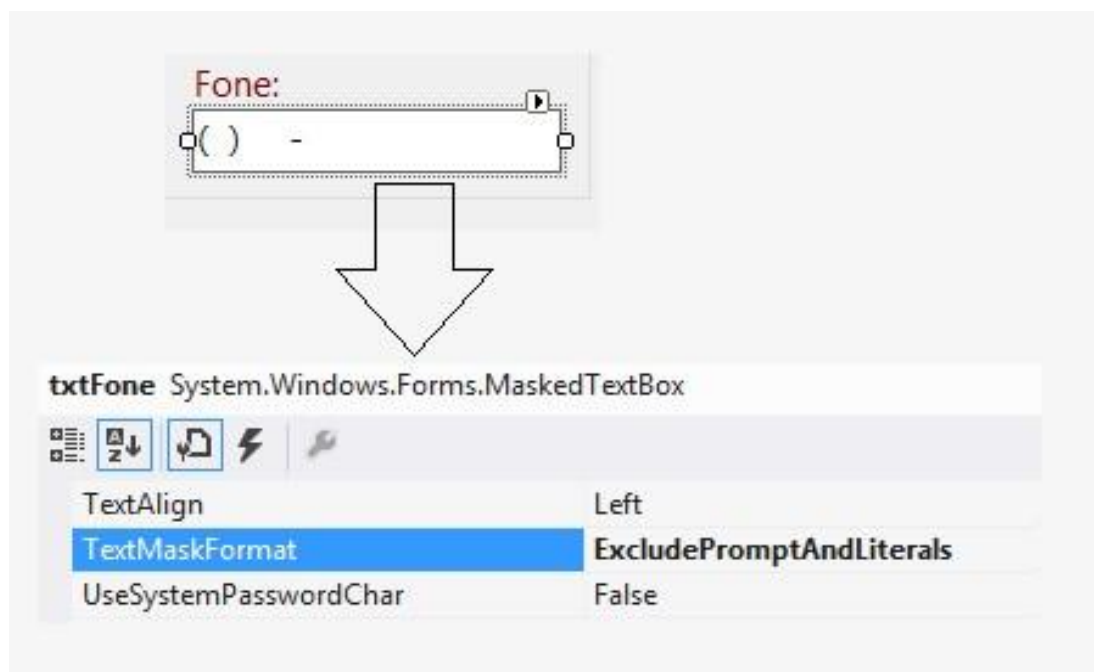


Teste o seu código:



Nome:	JORGE ANTUNES	Sala:	2J	gravar
Nascimento:	10/02/1996	Curso:	informatica	cancelar
Endereço:				
Rua:	PRF JOAO SANTANA	Número:	900_	
Bairro:	VL BOM JESUS	CEP:	18400-640	
Cidade:	ITAPEVA	Estado:	SP	
Email:	jo_an@uol.com.br	Fone:	(15)45632-3457	

Cuidado! Você receberá um a mensagem de erro caso digite campos com tamanhos maiores que definidos na tabela. Este problema irá ocorrer principalmente em CEP e Fone, caso você inclua os parentes e traços. Antes de executar o programa verifique se o parâmetro para não incluir os literais da máscara encontra-se selecionado, tanto para CEP quanto para o telefone:



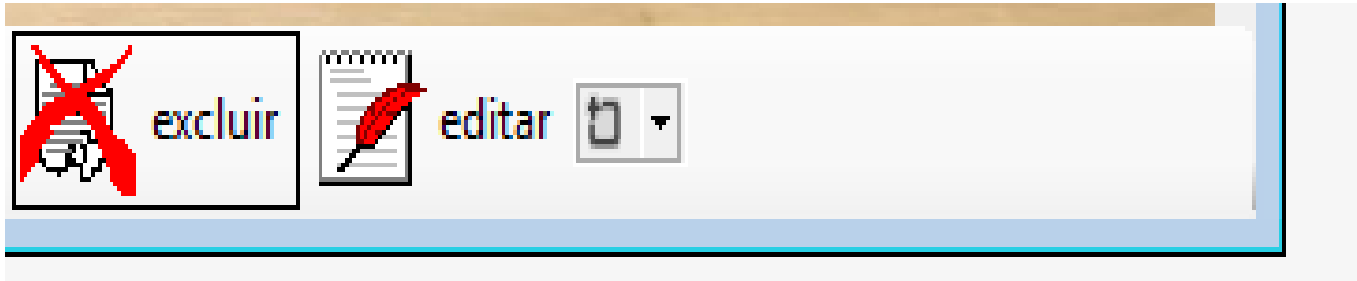
Também verifique a propriedade `maxlength` de todas as caixas de texto para que o valor seja menor que os valores definidos no valores `varchar ()` da tabela. Se tudo der certo você terá um novo registro cadastrado no banco de dados conforme a imagem a seguir:

Ficha Aluno:

Matrícula:	2	Série:	2J
Nome do Aluno:	JORGE ANTUNES		
Nascimento:	10/02/1996	Telefone:	154563234
Cidade:	ITAPEVA	Curso:	informatica

Exclusão de Registros

A exclusão é tecnicamente simples de ser realizada. Para isso codifique o evento click do botão excluir em FormAluno:



```
private void btDel_Click(object sender, EventArgs e)
{
    // não tem registros cadastrados?

    if (bn.PositionItem.Text.Equals("0"))
    {
        MessageBox.Show("Cadastro Vazio", "Mensagem",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    // conecta a tabela aluno
    controle.AlunoDB aDb = new controle.AlunoDB();
    // mensagem de exclusao
    if (MessageBox.Show("Remover " + lbAluno.Text,
        "Sistema", MessageBoxButtons.YesNo,
        MessageBoxIcon.Question) == DialogResult.Yes)
    {
        // pega o registro atual
        modelo.aluno reg = (modelo.aluno) bs.Current;
        // exclui do banco de dados
        aDb.excluir(reg);
        // remove da lista o registro
        bn.BindingSource.RemoveCurrent();
    }
}
```

Codificaremos o método excluir em **AlunoDB**:

```

public void excluir(modelo.aluno reg)
{
    using (var banco = new modelo.bibliotecaEntidades())
    {
        // conecta ao banco
        banco.Database.Connection.ConnectionString = con.open();
        // seleciona o registro a ser deletado usando o método single
        modelo.aluno aluno = banco.aluno.Single(qr => qr.idaluno == reg.idaluno);
        // remove o registro selecionado
        banco.aluno.Remove(aluno);
        // salva a informação no banco de dados
        banco.SaveChanges();
    }
}

```

A exclusão está pronta, selecione-se o código, enviando para a classe o aluno então é excluído, sem usar nenhuma instrução SQL.

Edição de Registros

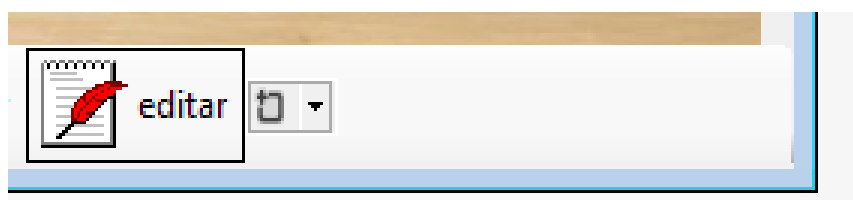
Para editar os dados de um registro será necessário localizar o registro, colocar os dados em um objeto aluno e em seguida enviar para o formulário. Começaremos codificando a rotina de pesquisa na classe:

```

public modelo.aluno procurar(modelo.aluno reg)
{
    using (var banco = new modelo.bibliotecaEntidades())
    {
        // conecta ao banco
        banco.Database.Connection.ConnectionString = con.open();
        // seleciona o registro a ser procurado usando o método single
        modelo.aluno aluno = banco.aluno.Single(qr => qr.idaluno == reg.idaluno);
        // retorna se encontrado
        return aluno;
    }
}

```

Codificar agora o botão de editar em **FormAluno**, gerando o evento click deste botão. Nele faremos a pesquisa e em seguida abriremos o formulário com os dados a serem preenchidos.



```

private void btEdit_Click(object sender, EventArgs e)
{
    if (bn.PositionItem.Text.Equals("0"))
    {
        MessageBox.Show("Cadastro Vazio", "Mensagem",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    FormDialogAluno fr = new FormDialogAluno();

    // registro atual
    modelo.aluno reg = (modelo.aluno)bs.Current;

    // conecta a tabela aluno
    controle.AlunoDB aDb = new controle.AlunoDB();
    // Retorna a pesquisa na própria variável 'reg'
    reg = aDb.procurar(reg);
    //Passa a variável reg para FormDialogAluno
    fr.Aluno = reg;
    fr.ShowDialog();
    // se atualizou
    if (fr.Aluno != null)
    {
        // reconstroi o bindingSource
        aDb.consultar(bs);
        bs.ResetBindings(false);
    }
}

```

No evento **Load** do formulário **FormDialogAluno** iremos preencher os campos com os valores encontrados pela pesquisa realizada. Isto se baseia no princípio que o botão de edição não envia o valor nulo para o formulário, isto permite preencher os dados do formulário com os valores encontrados pela pesquisa do sistema.

```

private void FormDialogAluno_Load(object sender, EventArgs e)
{
    // conecta a tabela curso
    controle.CursoDB cDB = new controle.CursoDB();
    // exibe os cursos no combobox
    cDB.listar(cbCurso);
    // se aluno é null então é novo
    if (Aluno == null) cbCurso.SelectedIndex = 0;
    else
    {
        // aluno não é null então editar
        //preenche os campos com os valores encontrados
        txtNome.Text = Aluno.nome;
        dtNascimento.Value = Aluno.nascimento;
        txtFone.Text = Aluno.fone;
        txtNumero.Text = Aluno.numero.ToString();
        cbUF.Text = Aluno.uf;
        txtBairro.Text = Aluno.bairro;
        txtEndereco.Text = Aluno.endereco;
        txtCidade.Text = Aluno.cidade;
        txtCEP.Text = Aluno.cep;
        txtEmail.Text = Aluno.email;
        txtSala.Text = Aluno.sala;
        cbCurso.SelectedValue = aluno.idcurso;
    }
}
}

```

Codificaremos o método editar responsável pela passagem de parâmetros para o banco de dados em nosso sistema. No formulário FormDialogAluno codifique o método editar:

```
private void editar()  
{  
    //Código aqui para editar o aluno  
    controle.AlunoDB aDB = new controle.AlunoDB();  
  
    Aluno.nome = txtNome.Text;  
    Aluno.nascimento = dtNascimento.Value;  
    Aluno.fone = txtFone.Text;  
    Aluno.numero = Convert.ToInt16(txtNumero.Text);  
    Aluno.uf = cbUF.Text;  
    Aluno.bairro = txtBairro.Text;  
    Aluno.endereco = txtEndereco.Text;  
    Aluno.cidade = txtCidade.Text;  
    Aluno.cep = txtCEP.Text;  
    Aluno.email = txtEmail.Text;  
    Aluno.sala = txtSala.Text;  
    Aluno.idcurso = Convert.ToInt16(cbCurso.SelectedValue);  
    aDB.editar(Aluno);  
}
```

Crie o método editar em AlunoDB:

```

private void FormDialogAluno_Load(object sender, EventArgs e)
{
    // conecta a tabela curso
    controle.CursorDB cDB = new controle.CursorDB();
    // exibe os cursos no combobox
    cDB.listar(cbCurso);
    // se aluno é null então é novo
    if (Aluno == null) cbCurso.SelectedIndex = 0;
    else
    {
        // aluno não é null então editar
        //preenche os campos com os valores encontrados
        txtNome.Text = Aluno.nome;
        dtNascimento.Value = Aluno.nascimento;
        txtFone.Text = Aluno.fone;
        txtNumero.Text = Aluno.numero.ToString();
        cbUF.Text = Aluno.uf;
        txtBairro.Text = Aluno.bairro;
        txtEndereco.Text = Aluno.endereco;
        txtCidade.Text = Aluno.cidade;
        txtCEP.Text = Aluno.cep;
        txtEmail.Text = Aluno.email;
        txtSala.Text = Aluno.sala;
        cbCurso.SelectedValue = aluno.idcurso;
    }
}

```

Pronto. A Rotina de edição no banco de dados encontra-se codificada. Programe apenas o botão cancelar para fechar o formulário e limpar o objeto aluno.

```

private void btCancel_Click(object sender, EventArgs e)
{
    Aluno = null;
    this.Dispose();
}

```

ROTINA DE PESQUISA

Vamos criar uma rotina de pesquisa para nosso formulário aluno. Para isso acrescente um botão de pesquisa no formulário aluno:

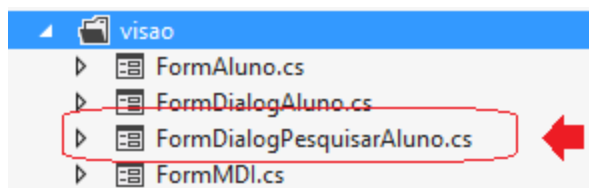


Chame o botão de **btnPesquisar**. Acrescente o seguinte código ao botão: (não se preocupe com as mensagens de erro, elas permanecerão até o formulário de pesquisa estiver criado).

```
private void btnPesquisar_Click(object sender, EventArgs e)
{
    // chama o formulário de pesquisa
    FormDialogPesquisarAluno fr = new FormDialogPesquisarAluno();
    fr.ShowDialog();

    if (fr.Id != 0) // se id do formulário seleciona corretamente um aluno
    {
        // acha a posição do registro na lista
        var obj = bs.List.OfType<modelo.aluno>().ToList().Find(
            linha => linha.idaluno == fr.Id
        );
        // passa a posição para o bindingsource bs
        int pos = bs.IndexOf(obj);
        // mostra os dados nos labels
        bs.Position = pos;
    }
}
```

Crie o formulário de pesquisa chamado **FormDialogPesquisarAluno**:



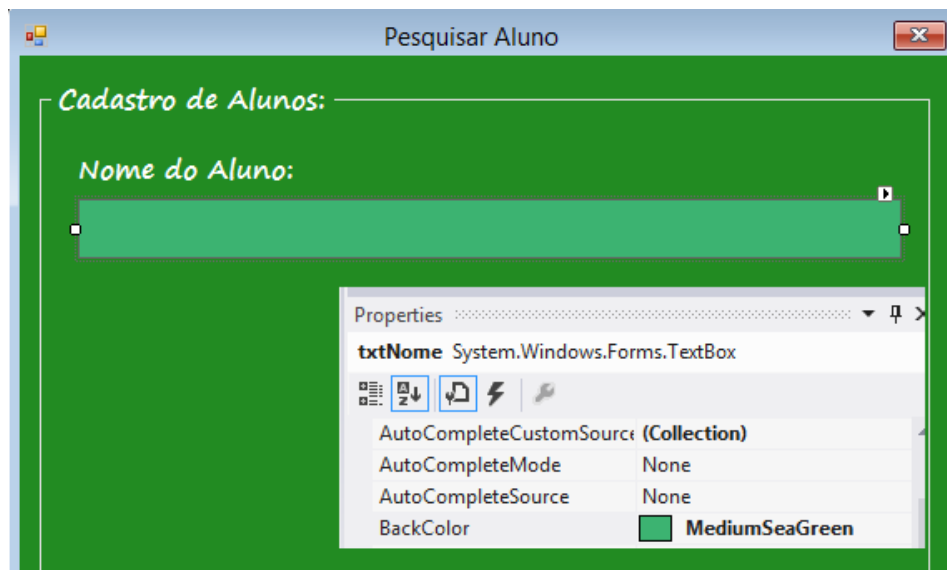
Acrescente ao início do formulário de pesquisa (tecla F7 no formulário) o campo Id que retornará o registro de aluno selecionado:


```
public partial class FormDialogPesquisarAluno : Form
{
    private int id;

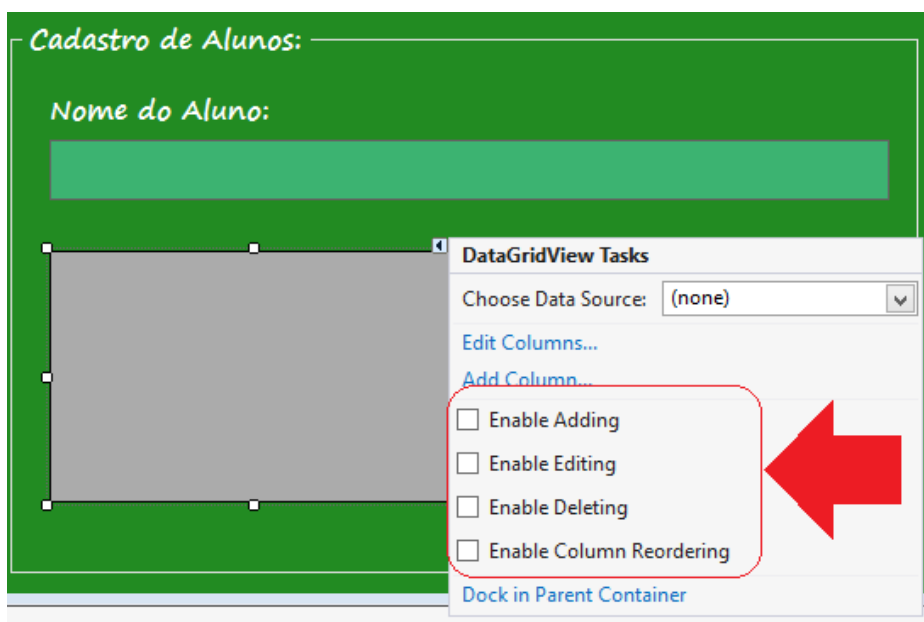
    public int Id
    {
        get { return id; }
        set { id = value; }
    }
}
```

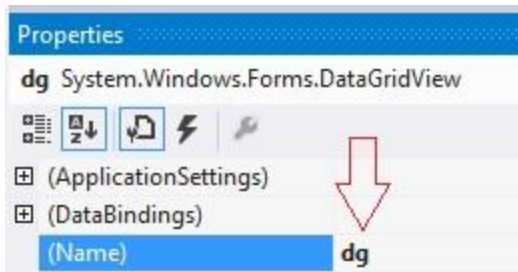


Desenhar uma caixa de texto chamada txtNome que será usado para procurar e identificar o nome do aluno:

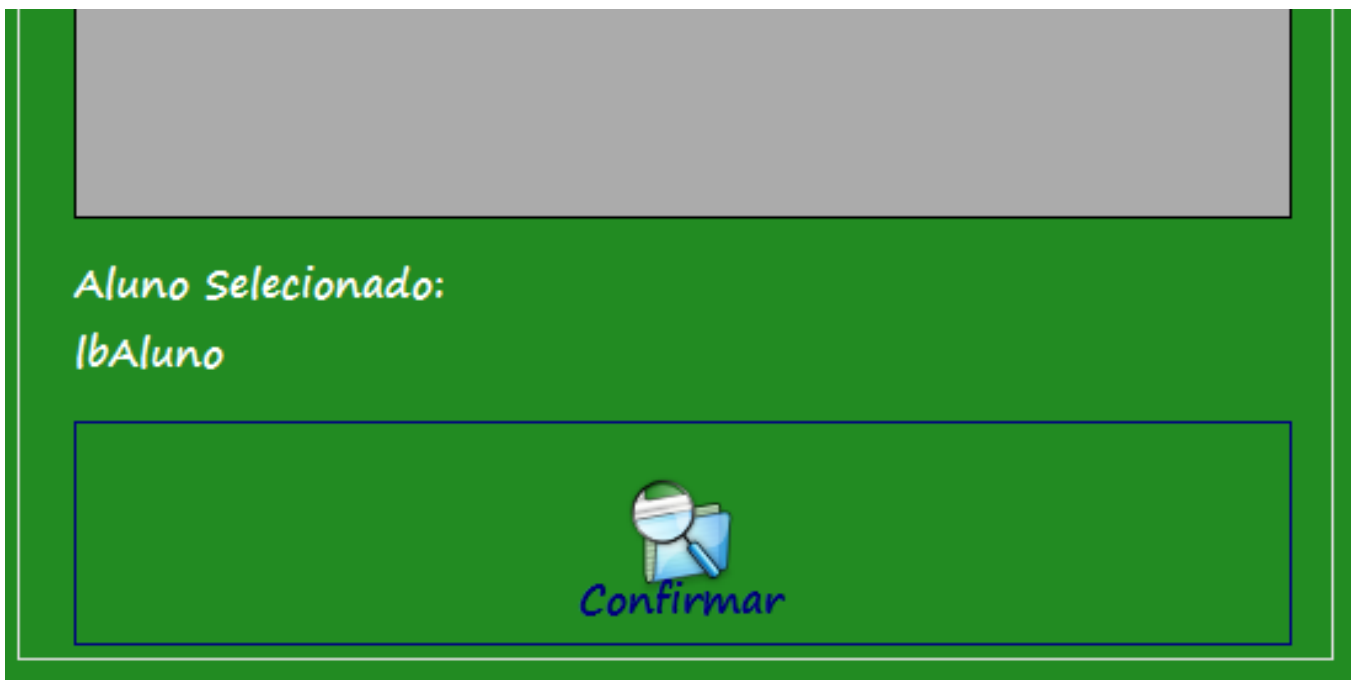


Em seguida desenhe um grid para exibir os nomes encontrados no banco de dados com as seguintes opções desmarcadas e Renomeio o grid para “dg”:





Agora acrescentar um botão chamado btOK e um label chamado lbAluno que será usado para confirmar o aluno selecionado.



Codificar o evento de pesquisa na caixa de texto, programando o evento `text_changed` da caixa de texto `txtNome`:

```
private void txtNome_TextChanged(object sender, EventArgs e)
{
    // conecta a tabela aluno
    controle.AlunoDB aDB = new controle.AlunoDB();
    // realiza a filtragem por nome
    aDB.filtrar(dg, txtNome.Text);
}
```

Crie o método de filtragem na classe `AlunoDB`:

```

public void filtrar(System.Windows.Forms.DataGridView dg, string p)
{
    using (var banco = new modelo.bibliotecaEntidades())
    {
        // conecta ao banco
        banco.Database.Connection.ConnectionString = con.open();
        // seleciona o registro a ser procurado usando o método single
        var reg = from linha in banco.aluno
                   where linha.nome.Contains(p)
                   orderby linha.nome
                   select new { linha.idaluno, linha.nome };
        // retorna se encontrado em dg

        dg.DataSource = reg.ToList();
        dg.Columns[1].Width = 500;
        // cores das colunas
        dg.Columns[0].DefaultCellStyle.ForeColor = System.Drawing.Color.Red;
        dg.Columns[1].DefaultCellStyle.ForeColor = System.Drawing.Color.Blue;
    }
}

```

Ao selecionar um item do grid, preencher lbAluno e a variável id:

```

private void dg_CellClick(object sender, DataGridViewCellEventArgs e)
{
    try
    {
        this.Id = Int16.Parse(dg.CurrentRow.Cells[0].Value.ToString());
        lbAluno.Text = dg.CurrentRow.Cells[1].Value.ToString();
    }
    catch (Exception)
    {
        this.Id = 0;
        txtNome.Text = "";
    }
}

```

Codificar o botão confirmar da pesquisa:

```
private void btOk_Click(object sender, EventArgs e)
{
    if (lbAluno.Text.Equals(""))
    {
        MessageBox.Show("Selecione um aluno válido");
    }
    else
        Dispose();
}
```

Teste o resultado:

Cadastro de Alunos:

Nome do Aluno:

ma

	<i>idaluno</i>	<i>nome</i>
▶	2	MARIA DO CARMO MACEDO

Aluno Selecionado:

MARIA DO CARMO MACEDO

Você pode modificar essa rotina para filtrar por mais itens além do nome. Modifique de acordo com as sua necessidade.

ROTINA DE INSERÇÃO DE CURSOS

Vamos modificar nosso formulário aluno para que ele também permita o cadastro de novos cursos dentro do projeto. Para garantir integridade não permitiremos exclusão de cursos que tenham alunos cadastrados

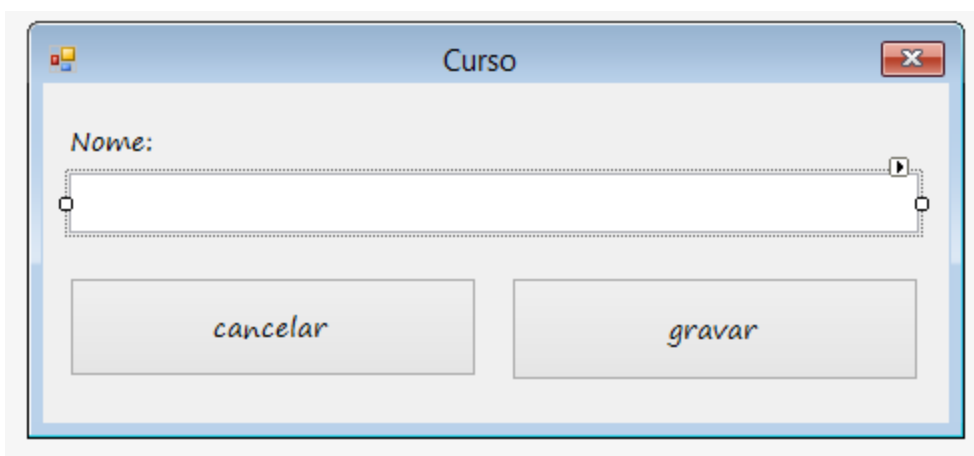
no sistema. Comece acrescentando um botão para adicionar e outro para remover cursos em FormDialogAluno, conforme a imagem a seguir, chamando-os de btAddCurso e btDelCurso:

A screenshot of a Windows Forms application. The form has several text boxes: 'Nome:', 'Sala:', 'Nascimento:' (containing '10/02/2016' and a calendar icon), 'Curso:', and 'Endereço:'. There are also two buttons: one with a green plus icon and another with a red minus icon. These two buttons are circled in red. The form is titled 'FormDialogAluno'.

Vamos chamar um formulário que irá cadastrar efetivamente um novo curso na lista:

```
private void btAddCurso_Click(object sender, EventArgs e)
{
    FormDialogCurso fr = new FormDialogCurso();
    fr.Curso = null;
    fr.ShowDialog();
    if (fr.Curso != null)
    {
        // conecta a tabela curso
        controle.CursoDB cDB = new controle.CursoDB();
        // exibe os cursos no combobox
        cDB.listar(cbCurso);
    }
}
```

Crie o formulário FormDialogCurso com a seguinte aparência:


A screenshot of a Windows Forms dialog box titled 'Curso'. It has a standard Windows title bar with a close button. Inside the dialog, there is a text box labeled 'Nome:'. Below the text box are two buttons: 'cancelar' and 'gravar'.

Nome dos componentes:

```
.TextBox txtNome  
.Label label1  
.Button btCancelar  
.Button btGravar
```

Acrescente um objeto da classe curso ao início do formulário:

```
public partial class FormDialogCurso : Form  
{  
    private modelo.curso curso;  
  
    public modelo.curso Curso  
    {  
        get { return curso; }  
        set { curso = value; }  
    }  
  
    public FormDialogCurso()  
    {  
        InitializeComponent();  
    }  
}
```



Codifique o evento “clique” para o botão cancelar:

```
private void btCancelar_Click(object sender, EventArgs e)  
{  
    Curso = null;  
    this.Dispose();  
}
```

Codifique o evento “clique” para o botão gravar, onde criaremos uma variável do tipo curso e iremos inserir os dados dentro da tabela curso, em caso de erro será exibido uma mensagem.

```

private void btGravar_Click(object sender, EventArgs e)
{
    try
    {
        if (Curso == null)
        {
            novo();
        }
        else
        {
            editar();
        }
        this.Dispose();
    }
    catch (EntityException err)
    {
        MessageBox.Show("ERRO DO EF:" + err.Message);
    }
}

```

Codificando o método **novo** em **FormDialogCurso**:

```

private void novo()
{
    controle.CursoDB cDB = new controle.CursoDB();
    Curso = new modelo.curso(){
        idcurso = cDB.proximoCodigo(),
        descricao = txtNome.Text.ToUpper()
    };
    cDB.inserir(Curso);
}

```

Vá para a classe CursoDB e codifique o método inserir:

```

public void inserir(modelo.curso novo)
{
    try
    {
        using (var banco = new modelo.bibliotecaEntidades())
        {
            // conecta ao banco
            banco.Database.Connection.ConnectionString = con.open();
            // adiciona o novo curso a lista
            banco.curso.Add(novo);
            // salva no banco de dados
            banco.SaveChanges();
        }
    }
    catch (Exception err)
    {
        // em caso de erro exibir a mensagem
        System.Windows.Forms.MessageBox.Show("Erro:" + err.Message);
    }
}

```

Codifique também o método que gera o próximo código em **CursoDB**:

```

public int proximoCodigo()
{
    int t = 0;
    try
    {
        using (var banco = new modelo.bibliotecaEntidades())
        {
            banco.Database.Connection.ConnectionString =
                con.open();
            var query = (from linha in banco.curso
                        select linha.idcurso).Max();
            t = Convert.ToInt16(query.ToString());
        }
        return t + 1;
    }
    catch (Exception err)
    {
        System.Console.WriteLine(err.Message);
        return 1;
    }
}

```


Desenvolver agora a rotina de exclusão, que será executada apenas se nenhum aluno estiver cadastrado no curso.

```
public void excluir(modelo.curso reg) {  
  
    using (var banco = new modelo.bibliotecaEntidades())  
    {  
        // conecta ao banco  
        banco.Database.Connection.ConnectionString = con.open();  
        // seleciona o registro a ser deletado usando o método single  
        modelo.curso curso = banco.curso.Single(qr => qr.idcurso == reg.idcurso);  
        // remove o registro selecionado  
        banco.curso.Remove(curso);  
        // salva a informação no banco de dados  
        banco.SaveChanges();  
    }  
}
```

É necessário checar se o curso não está sendo usado por um aluno antes de tentar a exclusão. Logo codifique o botão btDelCurso da seguinte forma em **FormDialogAluno**:

```
private void btDelCurso_Click(object sender, EventArgs e)  
{  
    controle.CursorDB cDb = new controle.CursorDB();  
    if (checarDependencia() == false)  
    {  
        //nenhum aluno usa o curso a ser excluido  
        modelo.curso reg = (modelo.curso)cbCurso.SelectedItem;  
        cDb.excluir(reg);  
        MessageBox.Show("Curso Excluido com sucesso");  
        // atualiza a lista de cursos  
        cDb.listar(cbCurso);  
    }  
    else  
        MessageBox.Show("Curso não pode ser excluido pois esta sendo usado");  
}
```

Desenvolver o método que checa a dependência

```
private bool checarDependencia()  
{  
    controle.AlunoDB aDB = new controle.AlunoDB();  
    // testa se curso já está sendo usado  
    return aDB.checarCurso(cbCurso.SelectedValue.ToString());  
}
```

Implemente o método checar dependência na classe AlunoDB:




```

public bool checarCurso(string p)
{
    using (var banco = new modelo.bibliotecaEntidades())
    {
        // conecta ao banco
        banco.Database.Connection.ConnectionString = con.open();
        // procura todos os alunos que tem curso
        int cod = Int16.Parse(p);
        var query = (from linha in banco.aluno
                     where linha.idcurso == cod
                     select linha).FirstOrDefault();
        if (query == null) return false;
        return true;
    }
}

```

Execute o programa:

Nome: Saia:

Nascimento:  Curso:  

Endereço:
Rua:
Bairro:
Cidade:

Curso não pode ser excluído pois está sendo usado

OK

Vamos codificar o método editar no formulário **FormDialogCurso**:

```

private void editar()
{
    controle.CursoDB cDB = new controle.CursoDB();
    Curso.descricao = txtNome.Text.ToUpper();
    cDB.editar(Curso);
}

```

Programando agora o evento **LOAD** no formulário **FormDialogCurso**:

```
private void FormDialogCurso_Load(object sender, EventArgs e)
{
    if (Curso != null) {
        txtNome.Text = Curso.descricao;
    }
}
```

Para a edição, vamos para o formulário FormDialogAluno e vamos programar o evento **KeyUp** do combobox **cbCurso** para executar a edição do curso:

```
private void cbCurso_KeyUp(object sender, KeyEventArgs e)
{
    // a tecla enter foi pressionada

    if (e.KeyCode == Keys.Enter) {
        FormDialogCurso fr = new FormDialogCurso();
        fr.Curso=(modelo.curso) cbCurso.SelectedItem;
        controle.CursoDB cDb = new controle.CursoDB();
        fr.ShowDialog();
        if (fr.Curso != null)
        {
            MessageBox.Show("Nome do curso Modificado com sucesso");
            // atualiza a lista de cursos
            cDb.listar(cbCurso);
        }
    }
}
```

Para encerrar codificar o método editar na classe CursoDB:

```
public void editar(modelo.curso reg)
{
    using (var banco = new modelo.bibliotecaEntidades())
    {
        banco.Database.Connection.ConnectionString = con.open();
        modelo.curso curso = banco.curso.Single(qr => qr.idcurso == reg.idcurso);

        curso.descricao = reg.descricao;
        banco.SaveChanges();
    }
}
```

Teste o código.

MONTANDO SISTEMA DE CEP EM C#

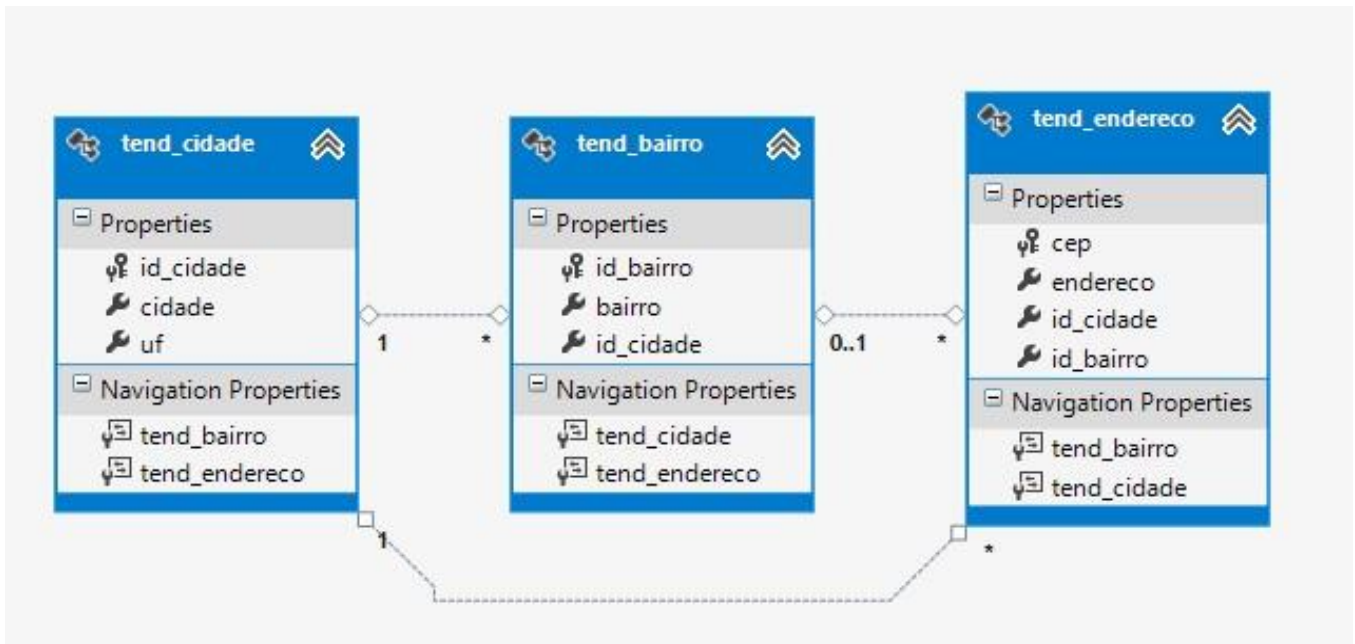
A base de dados de CEP contida no arquivo SQL fornecido, sendo composta por 3 tabelas:

tend_bairro

tend_cidade

ten_endereco

A estrutura das tabelas de acordo com o arquivo EDMX:

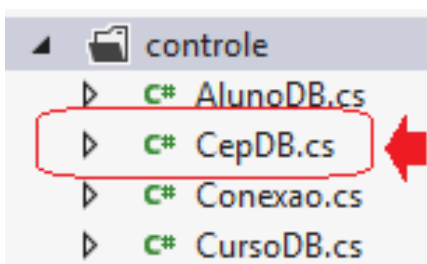


Acrescente um botão de pesquisa de CEP chamada **btCEP** no formulário **FormDialogAluno**:

O formulário de pesquisa de CEP contém os seguintes elementos:

- Campos de entrada para **Número:** e **CEP:**.
- Campos de entrada para **Estado:**.
- Um botão de pesquisa de CEP com um ícone de envelope e o texto **CEP**.

Para usar as tabelas vamos criar uma classe chamada CepDB:



Vamos codificar um método que procura o Código do CEP e devolve os dados do endereço:

```
class CepDB
{

    // objeto de conexão a base de dados

    Conexao con = new Conexao("localhost", "biblioteca",
        "root", "minas");

    public modelo.tend_endereco consultar(string cep)
    {
        // simplifica a chamada a biblioteca entidades
        using (var banco = new modelo.bibliotecaEntidades())
        {
            // conecta no banco de dados
            banco.Database.Connection.ConnectionString = con.open();
            // consulta usando a linguagem LINQ

            var query = (from linha in banco.tend_endereco.
                Include("tend_bairro").Include("tend_cidade")
                where linha.cep.Equals(cep)
                select linha).FirstOrDefault();

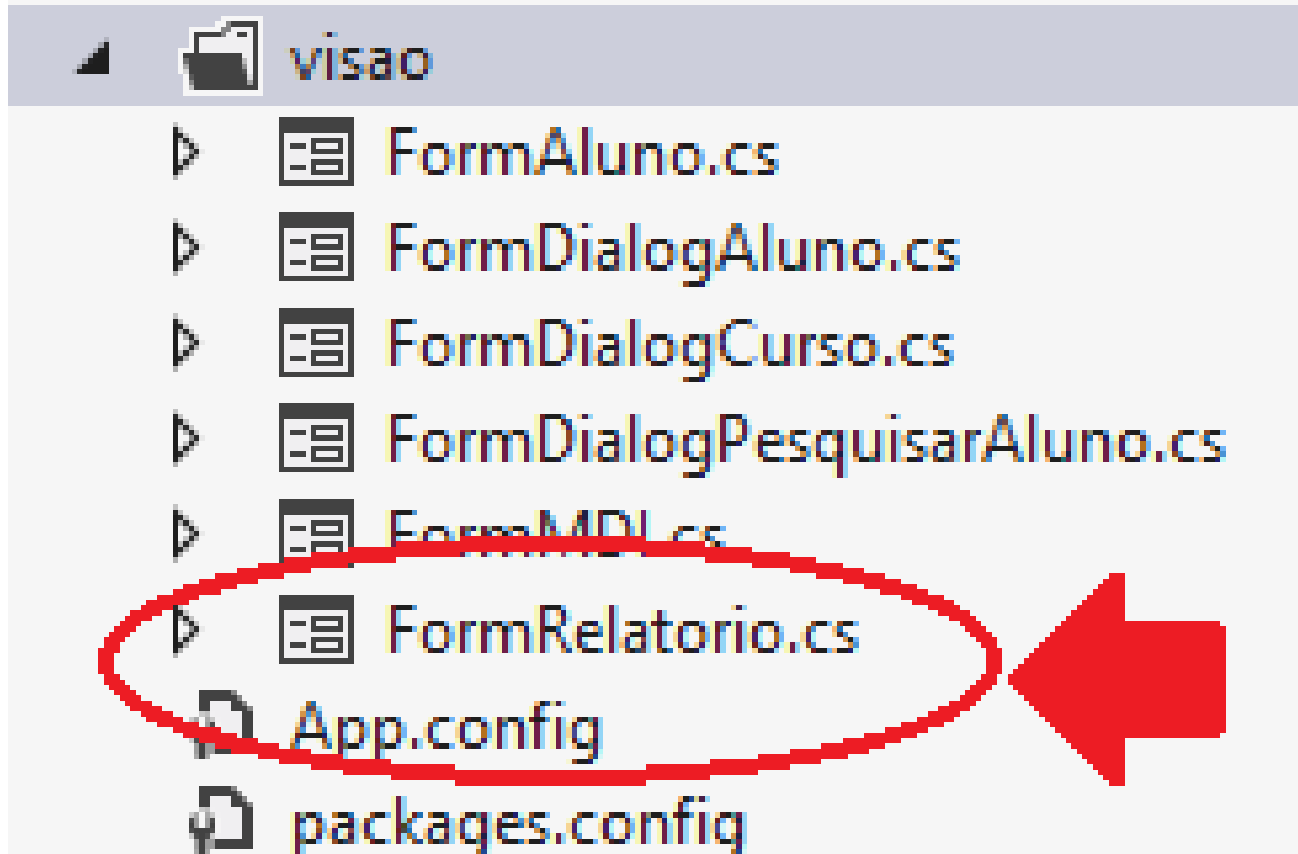
            if (query == null) return null;
            else
                return (modelo.tend_endereco)query;
        }
    }
}
```

Agora passemos a codificação do botão btCEP que será responsável por chamar a pesquisa de CEP em nossa base de dados.

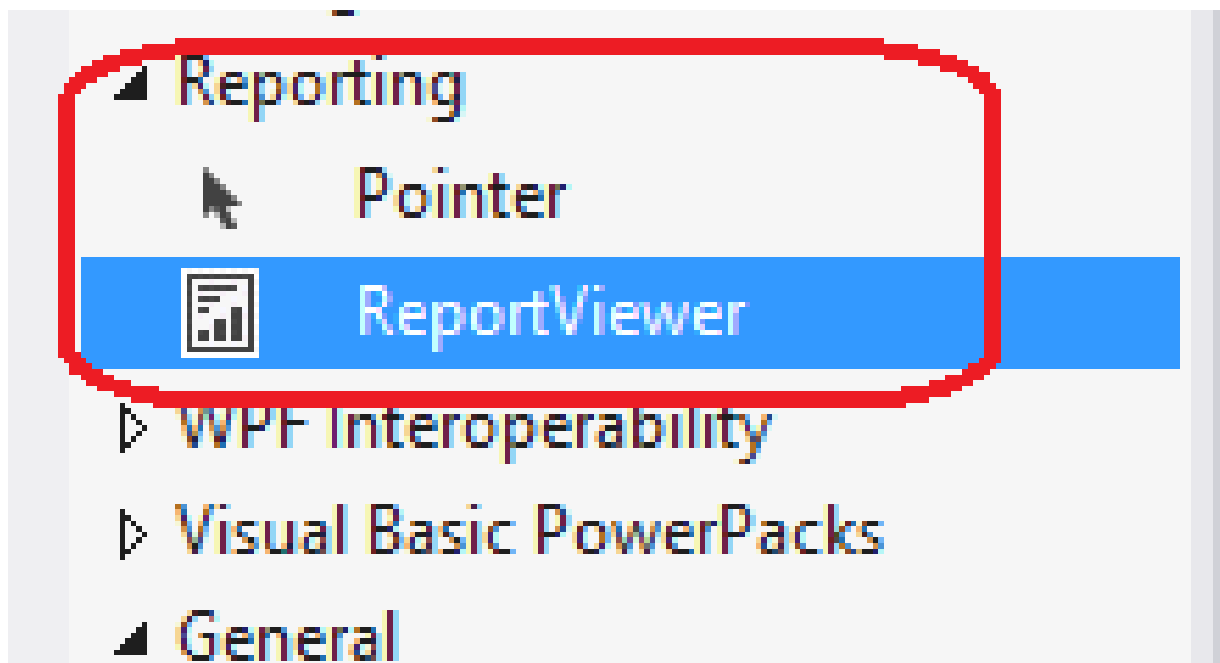
```
private void btCep_Click(object sender, EventArgs e)
{
    // conecta a base de dados de CEP
    controle.CepDB lista = new controle.CepDB();
    // retorna o endereço pesquisado
    modelo.tend_endereco endereco = lista.consultar(txtCEP.Text);
    if (endereco != null) {
        // preenche o endereço
        txtEndereco.Text = endereco.endereco.ToUpper();
        txtBairro.Text = endereco.tend_bairro.bairro.ToUpper();
        txtCidade.Text = endereco.tend_cidade.cidade.ToUpper();
        cbUF.SelectedItem = endereco.tend_cidade.uf.ToUpper();
        txtNumero.Focus();
    }
}
```

IMPRESSÃO – RELATÓRIOS

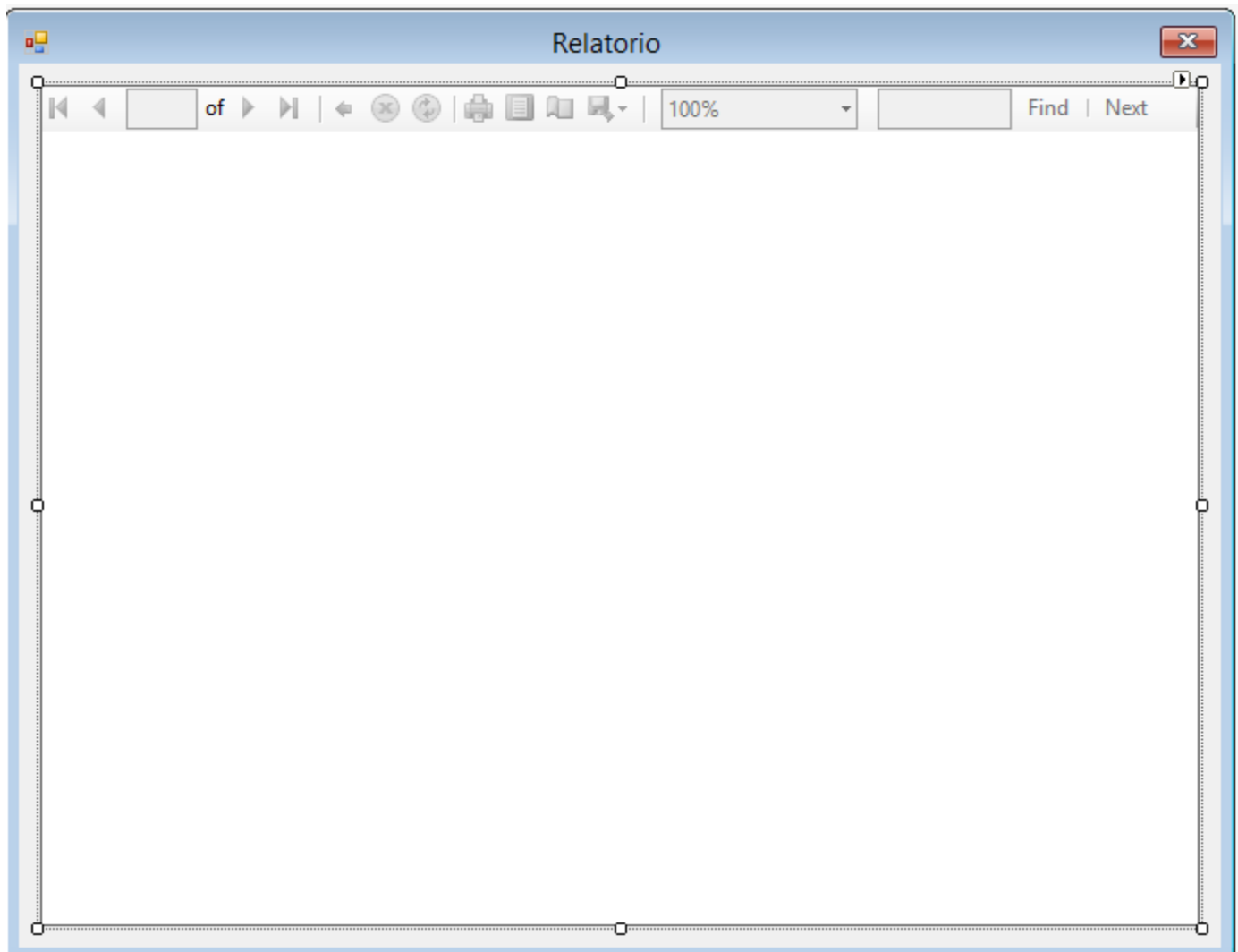
Vamos iniciar criando o formulário de exibição de relatórios:



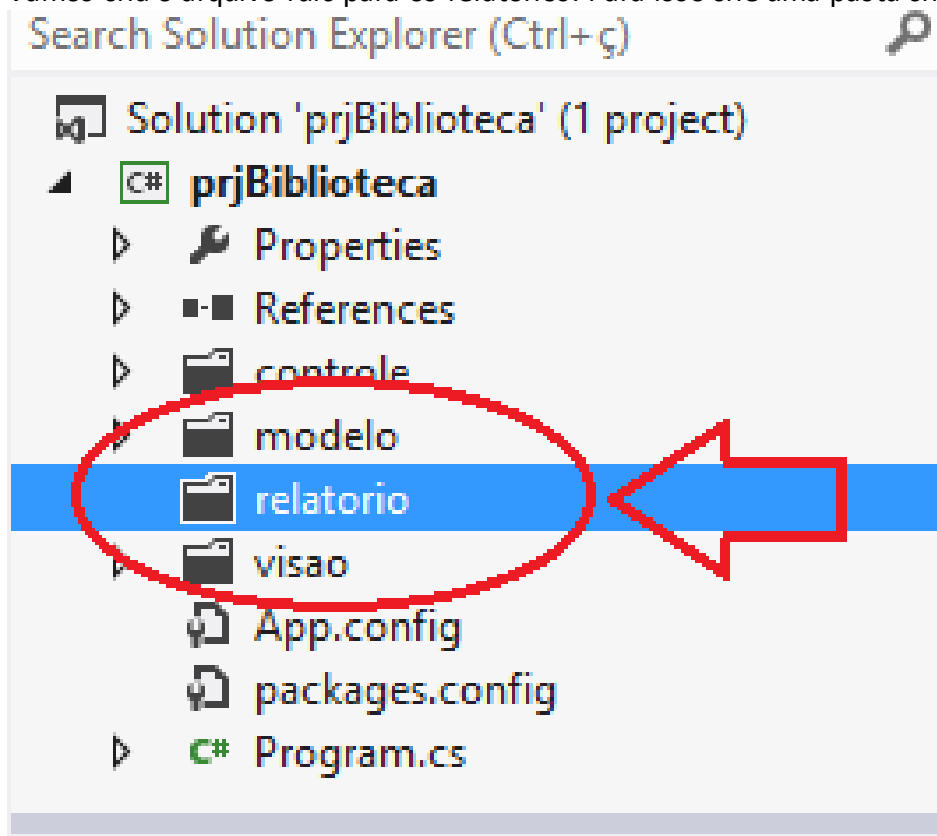
Insira neste formulário o componente relatório:



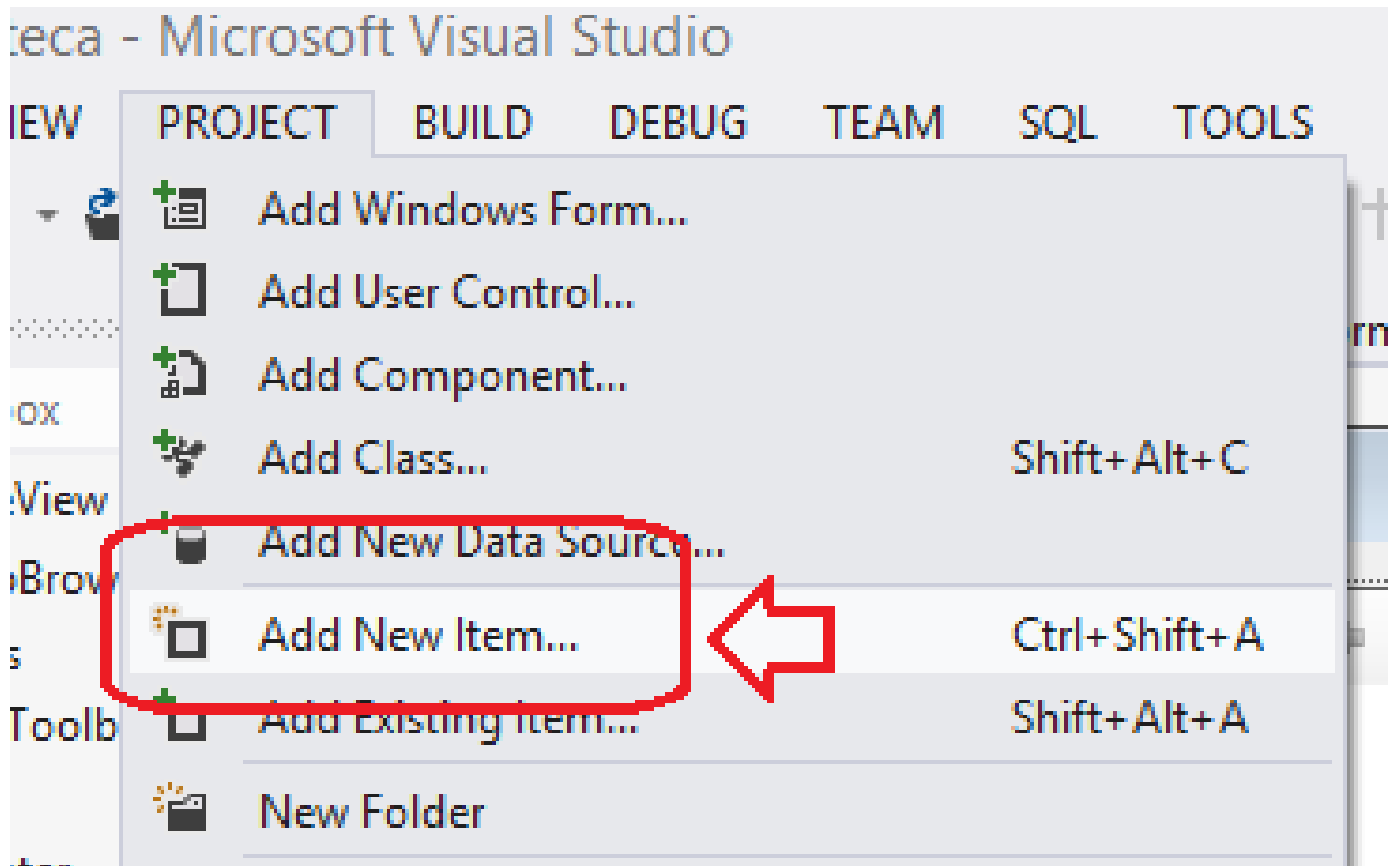
O Formulário vai ficar com a seguinte aparência:



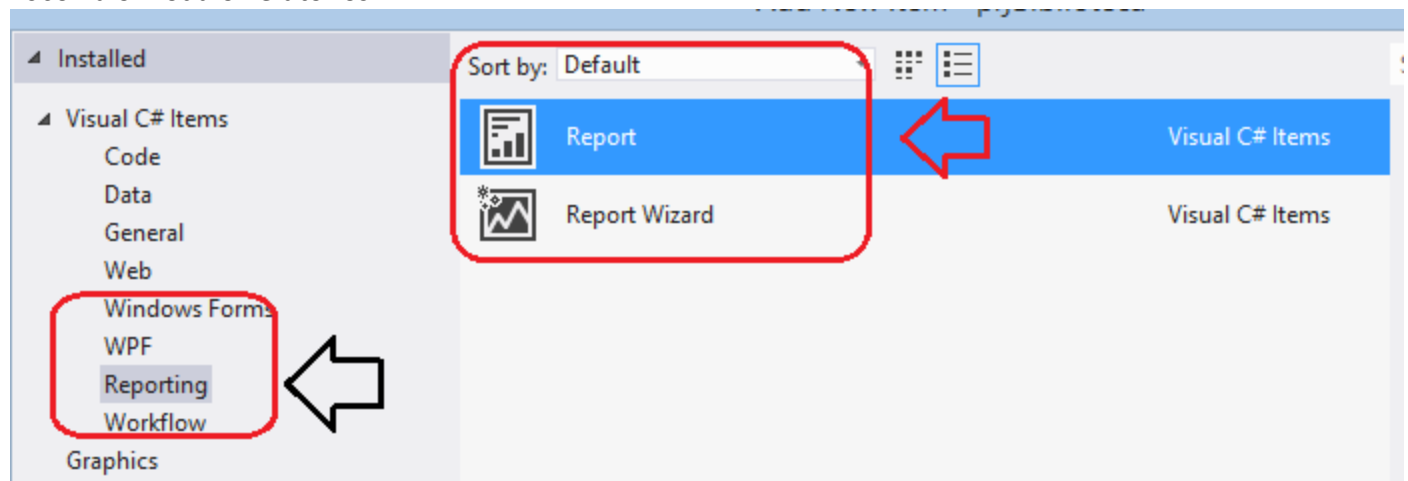
Vamos criar o arquivo rdlc para os relatórios. Para isso crie uma pasta chamada relatórios:



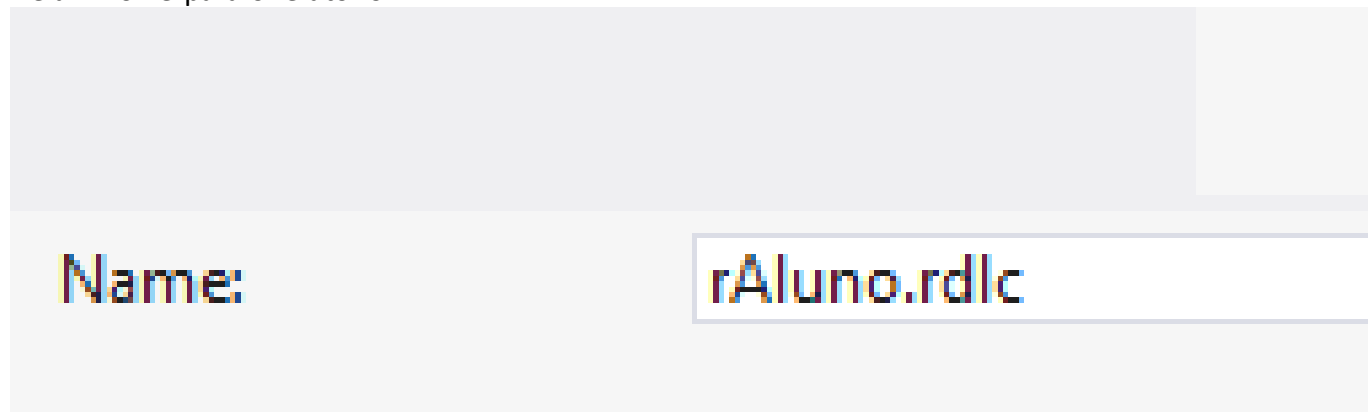
Vamos adicionar nela um arquivo rdlc de relatório usando o menu projeto:



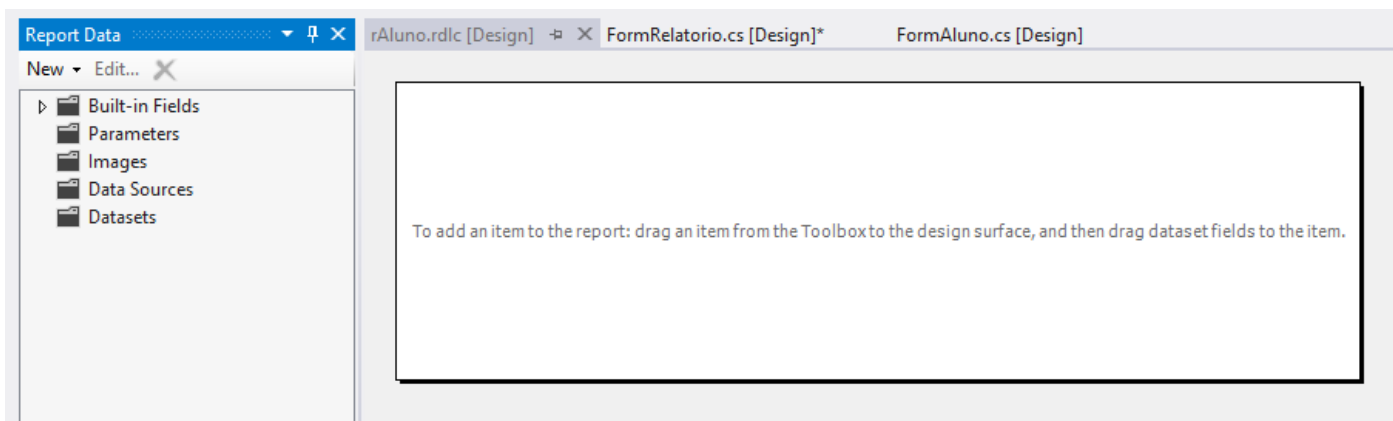
Escolha o módulo relatórios:



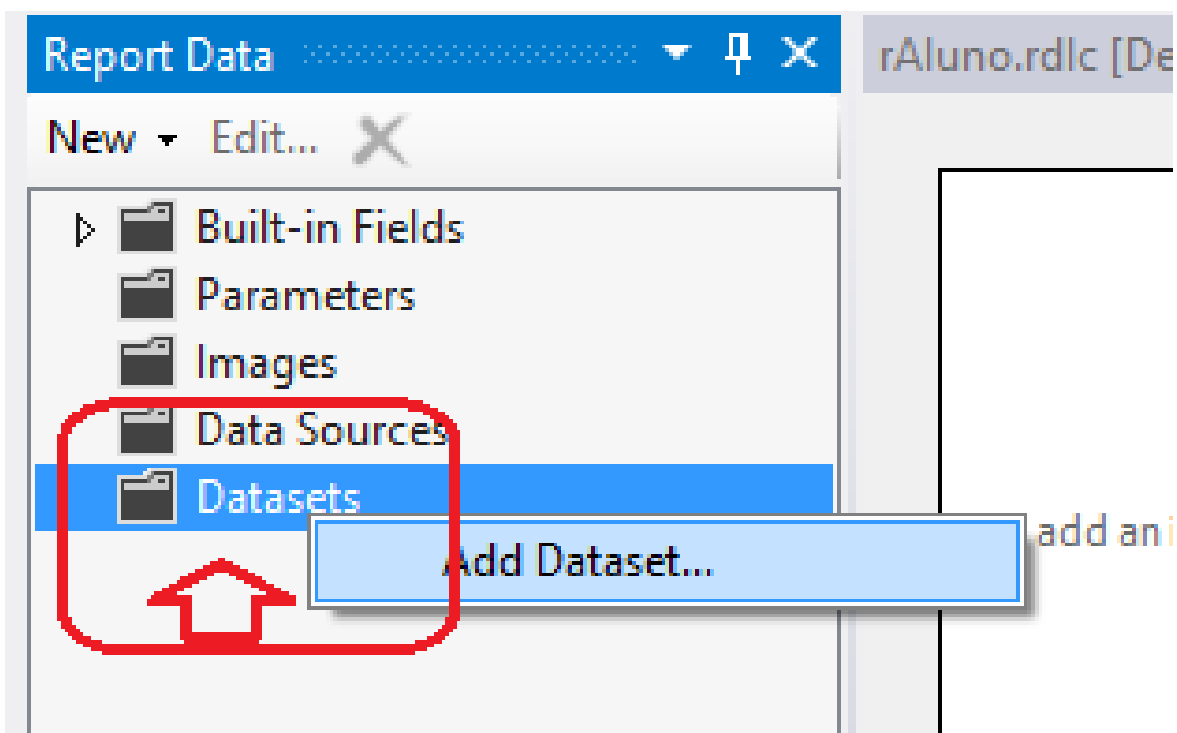
De um nome para o relatório:



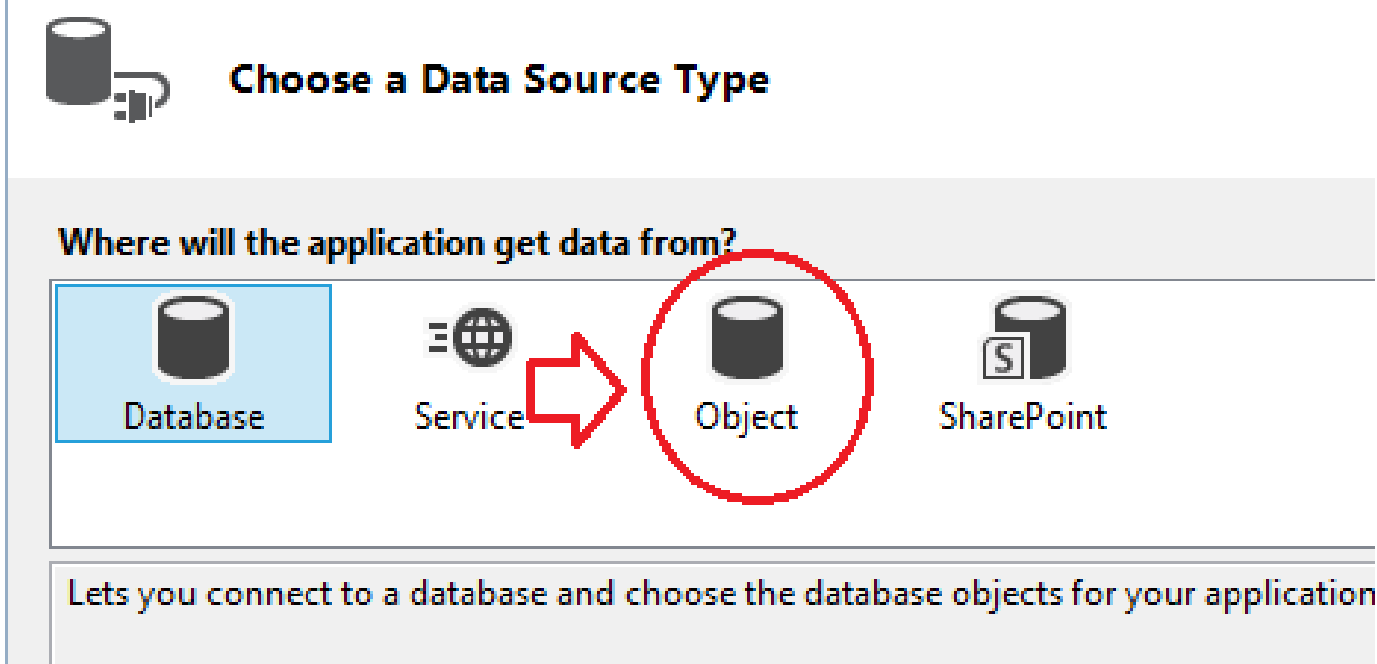
Pressione o botão adicionar (ou ADD em inglês). Você poderá agora iniciar o desenho de seu relatório:



A primeira coisa será criar um dataSet que usaremos para desenhar os campos a serem exibidos pelo relatório. Em dataSets com o botão direito escolha a opção “add dataSet”.



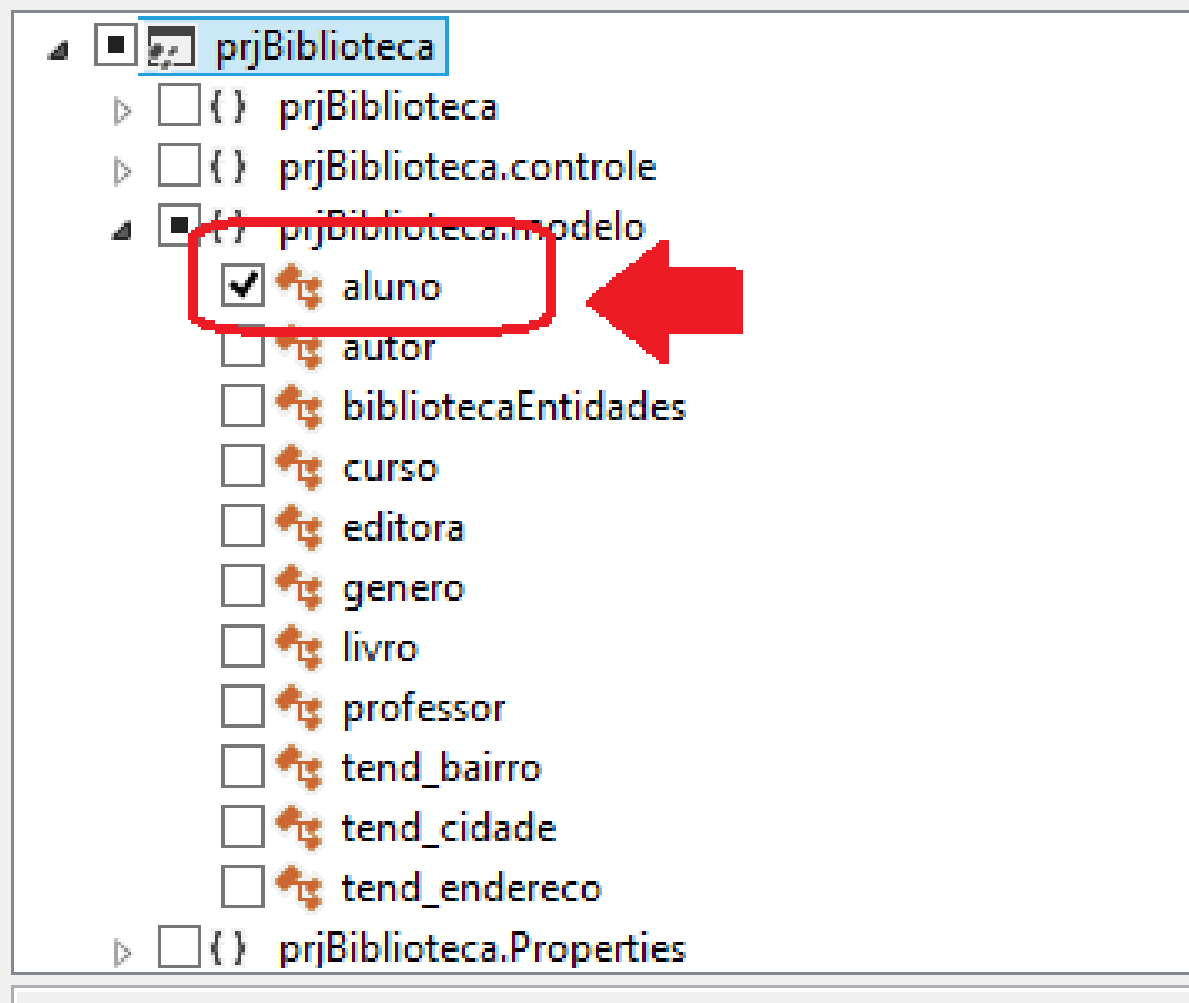
Selecione agora object:



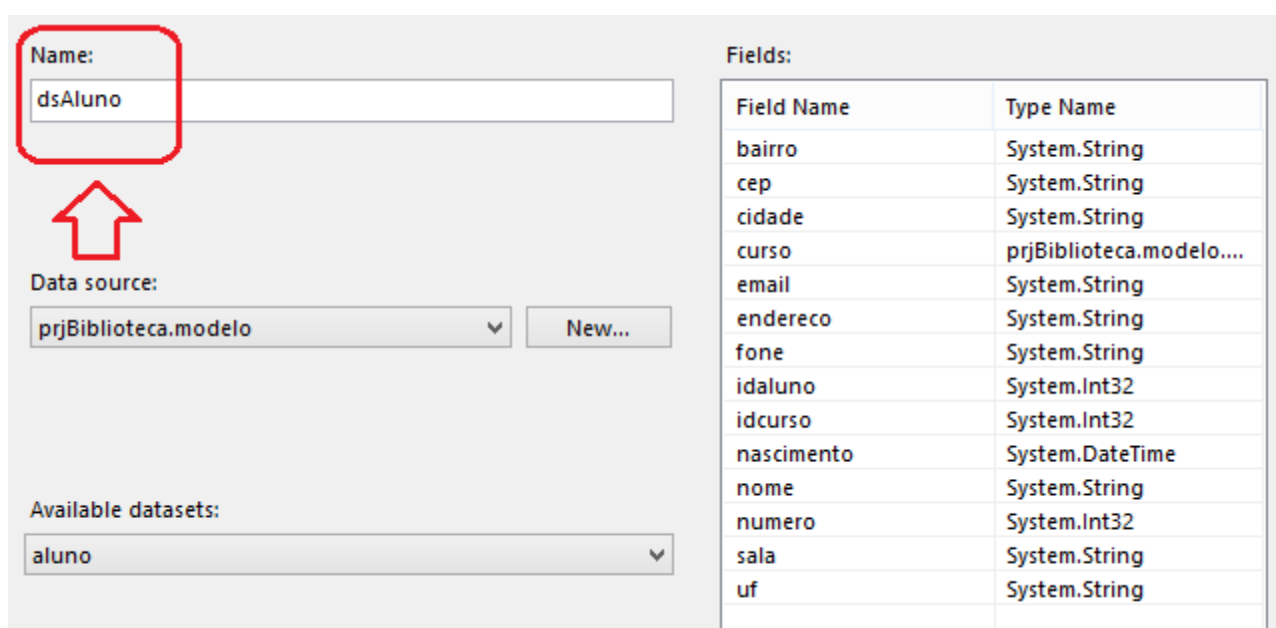
Navegue até encontrar o modelo aluno e selecione-o e em seguida pressione o botão finalizar (finish)

Expand the referenced assemblies and namespaces to select your ol assembly, cancel the wizard and rebuild the project that contains th

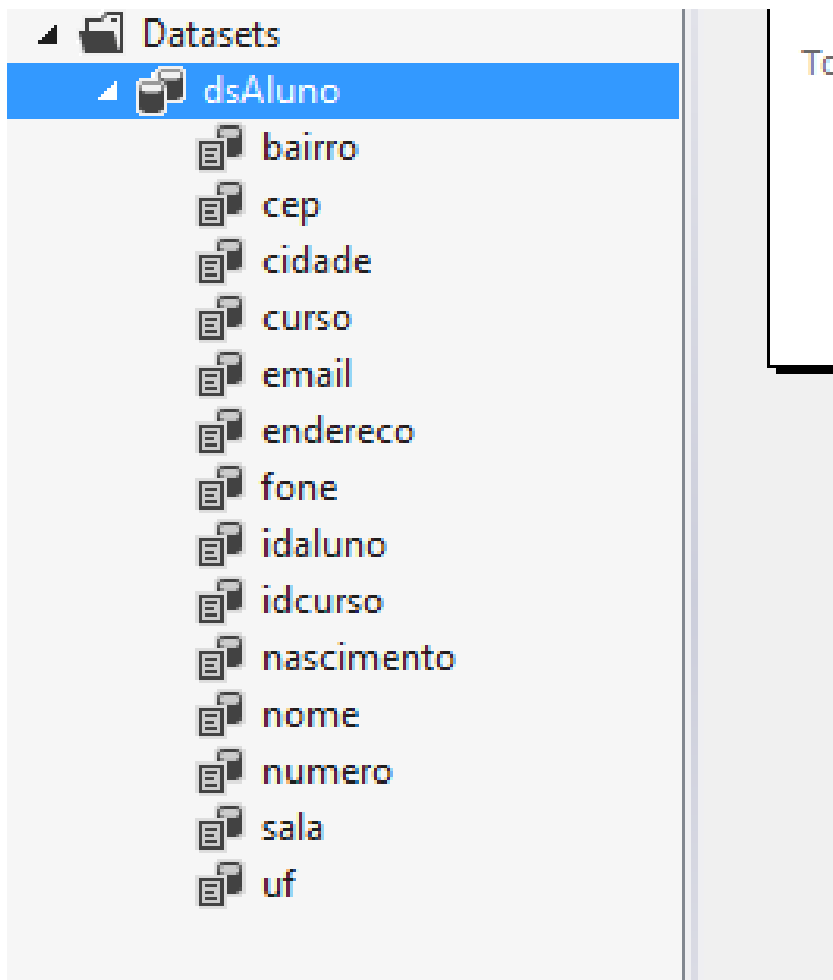
What objects do you want to bind to?



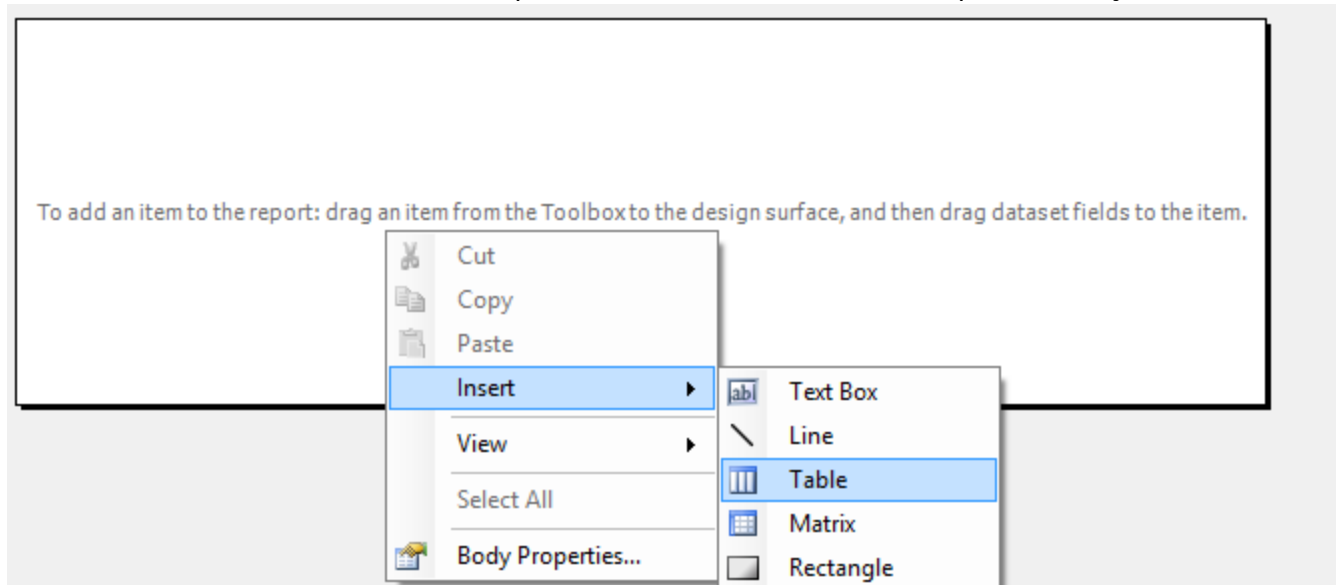
Renomeie o dataSet conforme a imagem e pressione o botão OK:



Agora temos um dataSet para desenhar o nosso relatório:



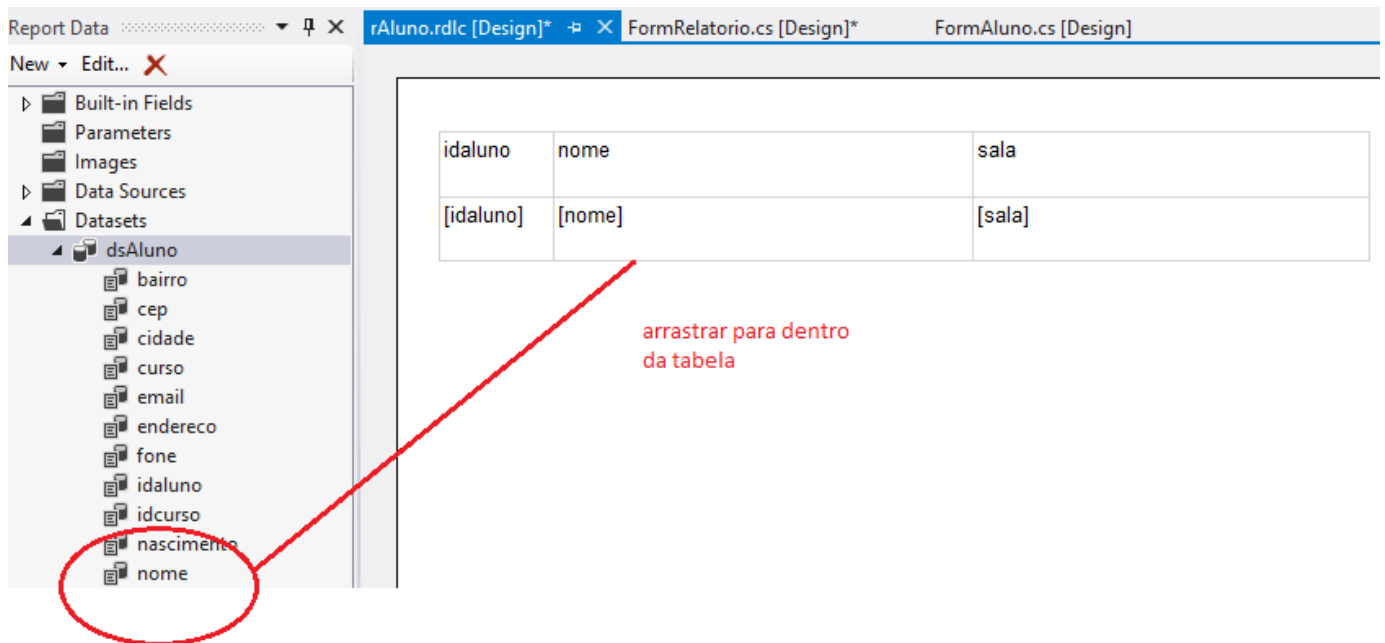
Para desenhar o relatório iniciaremos o processo adicionando uma tabela para a exibição dos dados:



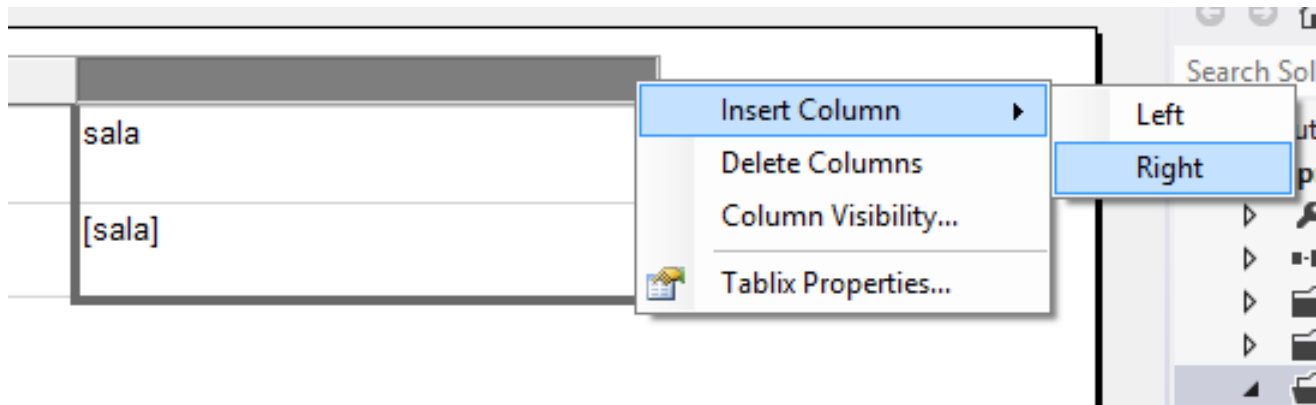
Teremos a seguinte visualização para os dados do relatório:

	Header	
	Data	

Vamos inserir os campos que vamos visualizar no relatório usando o dataSet:



Caso necessite adicione outras colunas a tabela a direita:



Adicione para o nosso exemplo uma coluna para conter o nome do curso:

idaluno	nome	sala	curso
[idaluno]	[nome]	[sala]	[curso]

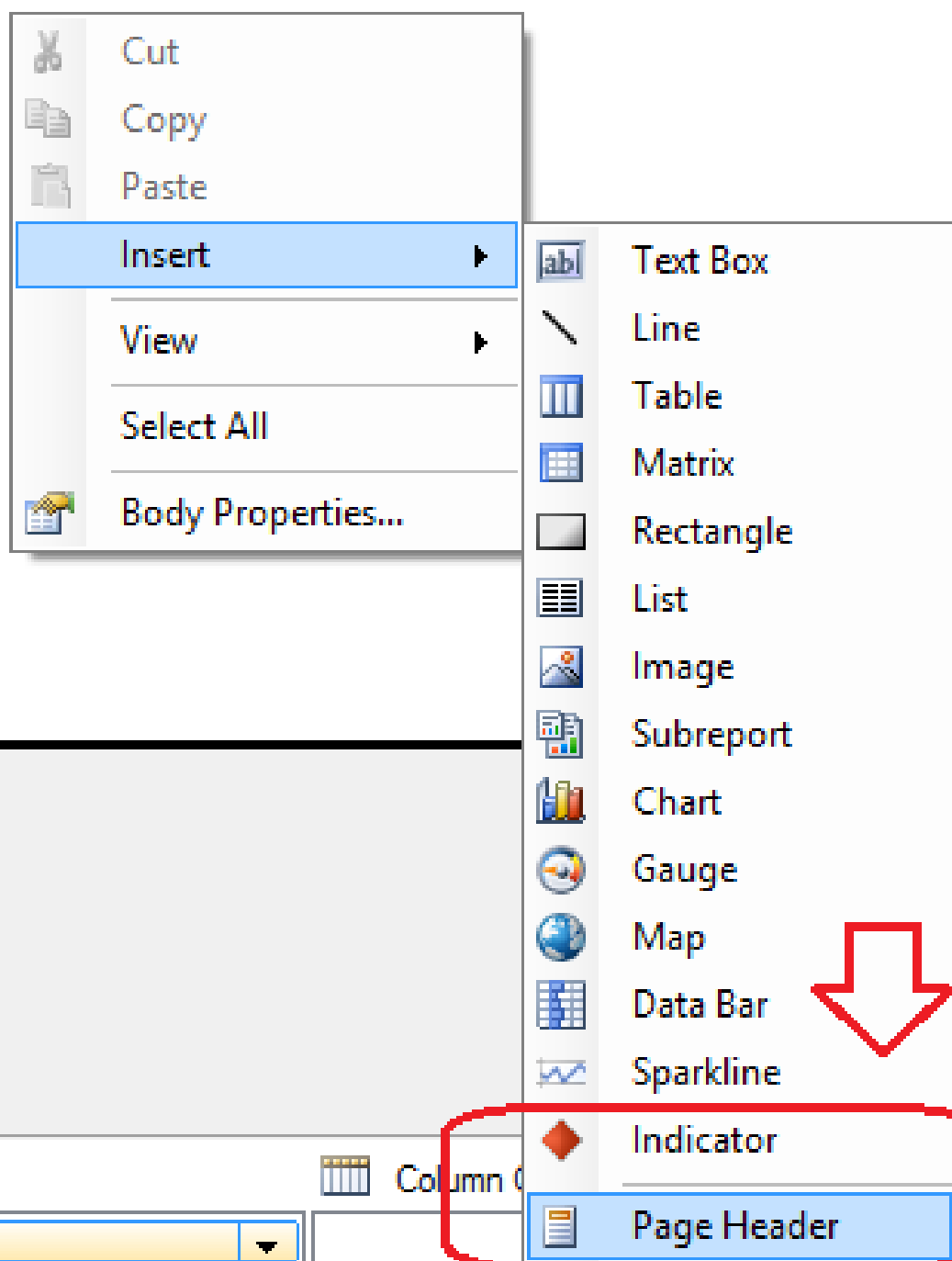
Em seguida edite os cabeçalhos dos campos da tabela para que fiquem com os seguintes valores:

Matrícula	Aluno	Série	Curso
[idaluno]	[nome]	[sala]	[curso]


Aproveite e mude as cores e o tipo de letra a serem usadas e define uma borda. O padrão de cores e letras pode ser alterado de acordo com as necessidades do relatório.

Matrícula	Aluno	Série	Curso
[idaluno]	[nome]	[sala]	[curso]

Acrescente agora um cabeçalho para o relatório, que irá aparecer em todas as páginas a serem impressas.



Desenhe o seguinte cabeçalho, com a inserção de um texto e de uma imagem para o logotipo do relatório.

 BIBLIOTECA - ETEC	RELATÓRIO DE ALUNOS CADASTRADOS DA BIBLIOTECA ETEC DR DEMETRIO AZEVEDO JUNIOR		
Matrícula	Aluno	Série	Curso
[idaluno]	[nome]	[sala]	[curso]

Agora temos o nosso relatório pronto. A etapa seguinte é projetar o formulário relatório para exibir o arquivo rdlc que acabamos de criar. Para isso crie o evento load para o formulário Relatório, codificando a estrutura básica do relatório:

```
private void FormRelatorio_Load(object sender, EventArgs e)
{
    try
    {
        // processa e exibe o relatório
        reportViewer1.ProcessingMode = ProcessingMode.Local;
        this.reportViewer1.RefreshReport();
    }
    catch (Exception erro)
    {
        MessageBox.Show(erro.Message, "Falha", MessageBoxButtons.OK, MessageBoxIcon.Error);
        this.Dispose();
        return;
    }
}
```

Precisamos definir agora que relatório será gerado e que dataSet será fonte para as informações. Crie o dataSet e o caminho para o relatório:

```
public partial class FormRelatorio : Form
{
    private string caminho;

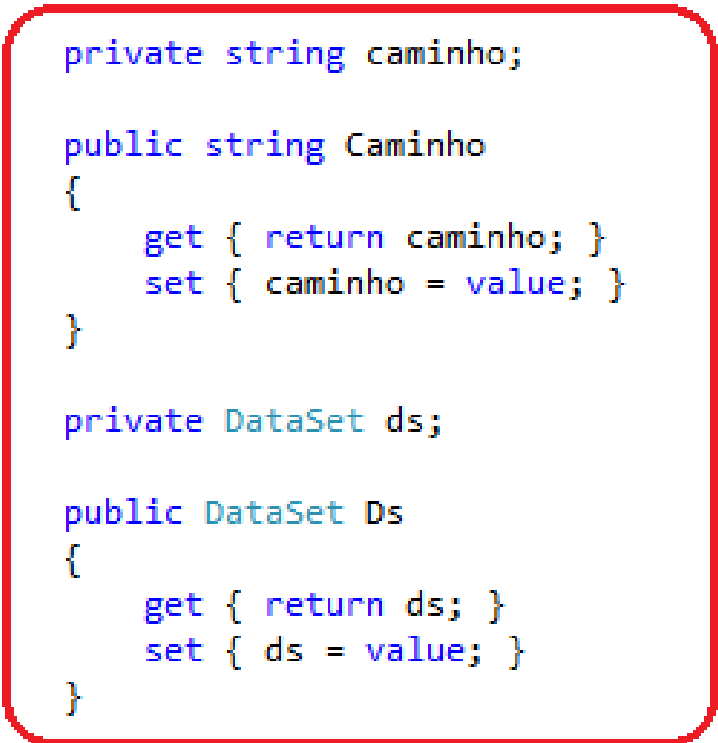
    public string Caminho
    {
        get { return caminho; }
        set { caminho = value; }
    }

    private DataSet ds;

    public DataSet Ds
    {
        get { return ds; }
        set { ds = value; }
    }

    public FormRelatorio()
    {
        InitializeComponent();
    }

    private void FormRelatorio_Load(object sender, EventArgs e)
    {
```



Vamos carregar o dataSet e o relatório para o formulário:

```
private void FormRelatorio_Load(object sender, EventArgs e)
{
    try
    {
        // limpa o relatório
        this.reportViewer1.LocalReport.DataSources.Clear();
        ReportDataSource source = null;

        //local do relatório
        this.reportViewer1.LocalReport.ReportPath = @caminho;

        // carga do dataSet
        source = new ReportDataSource(ds.DataSetName, ds.Tables[0]);
        // prepara o relatório para exibição
        this.reportViewer1.LocalReport.DataSources.Add(source);
        this.reportViewer1.DocumentMapCollapsed = false;
        // processa e exibe o relatório
        reportViewer1.ProcessingMode = ProcessingMode.Local;
        this.reportViewer1.RefreshReport();
    }
    catch (Exception erro)
    {
        MessageBox.Show(erro.Message, "Falha", MessageBoxButtons.OK, MessageBoxIcon.Error);
        this.Dispose();
        return;
    }
}
}
```

Em FormAluno vamos criar um botão para a geração do relatório e chame de **btnRelatorio**



Crie o evento click para esse botão:

```
private void btnRelatorio_Click(object sender, EventArgs e)
{
    DataSet ds = new DataSet();
    // conecta a tabela aluno
    controle.AlunoDB aDB = new controle.AlunoDB();
    aDB.imprimir(ds); // rotina que monta o dataset
    FormRelatorio fr = new FormRelatorio();
    // define que vou imprimir dataSet de aluno
    fr.Ds = ds;
    // local do relatório
    fr.Caminho = "D:\\projetos\\prjBiblioteca\\prjBiblioteca\\relatorio\\rAluno.rdlc";
    // exibe o relatório
    fr.ShowDialog();
}
}
```


Agora é só codificar a rotina de montagem para os campos do dataSet contidas no método imprimir de AlunoDB:

```

public void imprimir(System.Data.DataSet ds)
{
    using (var banco = new modelo.bibliotecaEntidades()) // conecta ao arquivo EDMX
    {
        banco.Database.Connection.ConnectionString = con.open();
        // realiza a consulta usando a linguagem LINQ
        var query = from linha in banco.aluno
                     orderby linha.nome
                     select new
                     {
                         linha.idaluno, linha.nome,
                         linha.sala, linha.cidade,
                         linha.curso.descricao
                     };
        ds.Tables.Add(new System.Data.DataTable());
        // as colunas devem bater com o nome usado no rdlc
        ds.Tables[0].Columns.Add("idaluno");
        ds.Tables[0].Columns.Add("nome");
        ds.Tables[0].Columns.Add("sala");
        ds.Tables[0].Columns.Add("cidade");
        ds.Tables[0].Columns.Add("curso");
        foreach (var a in query)
        {
            System.Data.DataRow row = ds.Tables[0].NewRow();
            row["idaluno"] = a.idaluno;
            row["nome"] = a.nome;
            row["sala"] = a.sala;
            row["cidade"] = a.cidade;
            row["curso"] = a.descricao;
            ds.Tables[0].Rows.Add(row);
        }
        ds.DataSetName = "dsAluno";
    }
}

```

Agora podemos testar o relatório:

Relatorio			
<div>  <div> RELATÓRIO DE ALUNOS CADASTRADOS DA BIBLIOTECA ETEC DR DEMETRIO AZEVEDO JUNIOR </div> </div>			
Matrícula	Aluno	Série	Curso
4	JOANA DA SILVA	2J	INFORMATICA
2	JOANA DOS SANTOS DIAS MONTEIRO	1E	INFORMATICA
1	JOAO CARLOS MARTINS	1J	INFORMATICA
3	MARCELA DOS SANTOS ANDRADE	1H	enfermagem