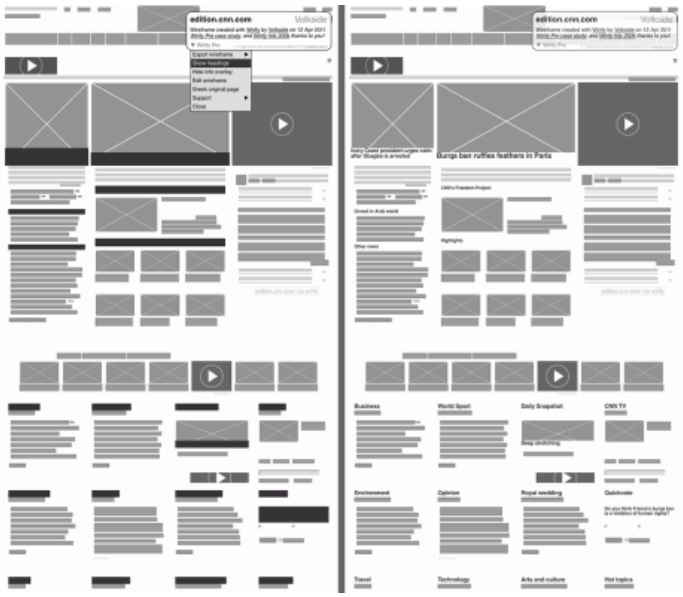
# *Projeto Interdisciplinar – Roteiro*

## Planejamento

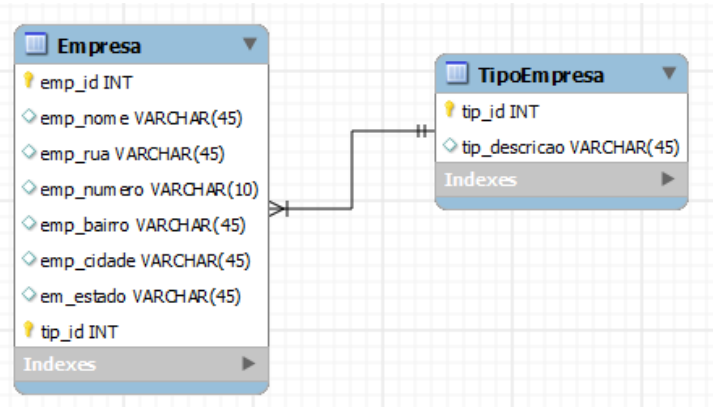
### Primeiro passo:

Levantamento de Requisitos do sistema.
Prototipação do sistema (wireframe).



### Segundo passo:

Modelagem e criação do Banco de Dados.
Iremos utilizar o exemplo abaixo, lembrando que este exemplo é meramente ilustrativo.
Observe as chaves primárias e a chave estrangeira.



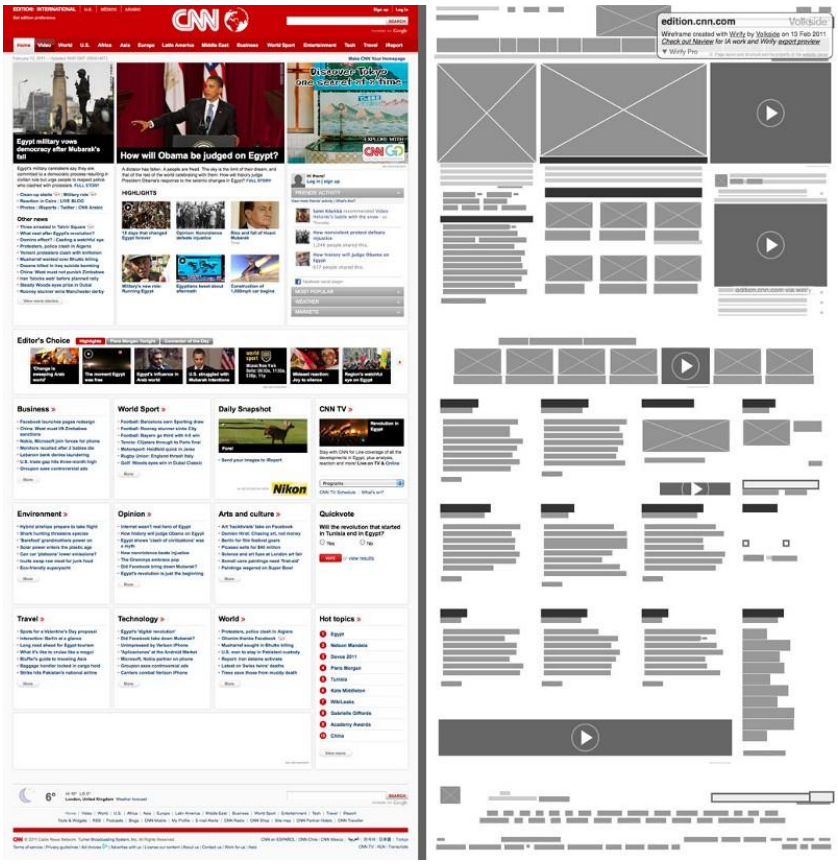

### Criando o exemplo acima:

```
Create database projeto;
use projeto;
```

```
create table  tip_tipoempresa(
    tip_id integer primary key auto_increment,
    tip_descricao varchar( 120 )
);


create table emp_empresa(
    emp_id integer primary key auto_increment,
    emp_nome varchar (120),
    emp_rua varchar (120),
    emp_numero integer,
    emp_bairro varchar (120),
    emp_cidade varchar (120),
    emp_estado varchar (5),
    tip_id integer,
    foreign key ( tip_id ) references tip_tipoempresa ( tip_id )
);
```

## Planejamento – Master Page (Layout)

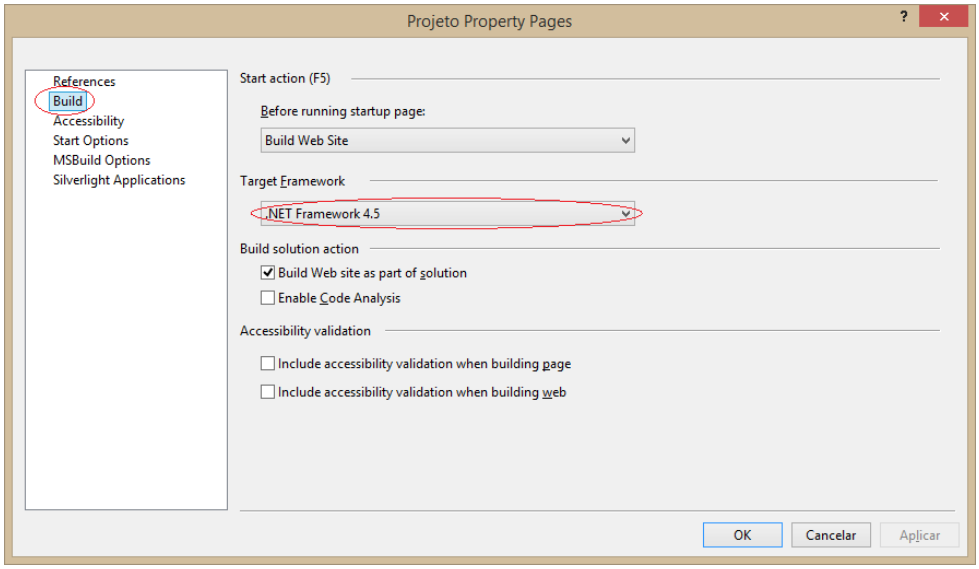
Definir o que será comum à todas as páginas e criar a Master Page (Layout).
Isso facilita a reutilização do código e a distribuição das tarefas entre os membros da equipe.



## Verificando a versão do .NET Framework

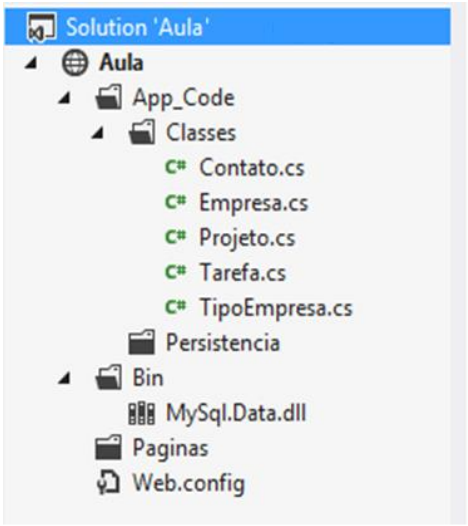
### Alterando a versão do .NET Framework

Botão direito sobre o projeto / Property Pages / Build
Em: “Target Framework” selecione a versão do .NET Framework desejado e clique em Ok.




## Definir o padrão de pastas e organizar os arquivos

Aqui verificamos um exemplo simples.
Novas pastas podem ser criadas de acordo com a necessidade.




### Definição

Pasta App_Code:

Armazenará todas as nossas classes, dentro da pasta App_Code criamos as pastas **Classe** e **Persistência.**

Pasta Classe:

As Classes são criadas de acordo com a modelagem do Banco de dados.
Atenção para os tipos de dados, eles não podem contrariar o que foi modelado no banco de dados.
Devemos criar uma estrutura básica das classes.

Pasta Persistência:

Armazenará as classes de persistências, que são as classes que trabalharão com o acesso ao banco de dados (Select, Delete, Insert, Update).
Nem toda classe precisa obrigatoriamente de uma classe de persistência.

Pasta Bin:

Contém todas os arquivos de configuração (dll do projeto – por exemplo **MySqlData**).

## dll MySqlData na pasta Bin

Este arquivo **MySqlData** é um arquivo de ponte, é ele quem fornece a conexão da aplicação com o banco de dados.
Sem ele não é possível acessar nossa base de dados.
Você pode copiar o arquivo e colar dentro desta pasta Bin.

Ou clique com o botão direito do mouse sobre a pasta Bin / Add References / Browser / Ache o local onde está o arquivo MySqlData.dll. / ok
Pode (ou não) ser exibido uma mensagem avisando que as configurações serão alteradas.

**Atenção:**

A pasta não pode ficar vazia, em algumas situações, mesmo executando a operação descrita acima a pasta permanece vazia. Se isso acontecer, localize o arquivo copie o cole o arquivo dentro da pasta.

Clique com o botão direito do mouse sobre a pasta Bin e dê um **refresh** para verificar se a pasta contém o arquivo MysqlData.dll.

# String de Conexão

### Configurar a String de Comexão no Web.Config

No Web.Config nós iremos criar nossa String de Conexão.

A String de conexão contém o nome do usuário do banco, senha do banco, onde o banco está armazenado e qual o nome do database que iremos utilizar.

Cada vez que a aplicação solicitar um serviço do banco de dados, ele vai validar esta String de conexão, se for válida ele acessa o banco de dados, executa suas operações e retorna essa validação.

## Script - String de Conexão

```
<appSettings>
    <add key= "strConexao" value="Database = projeto; Data Source = localhost; User id = root;
    Password=; pooling = false; "/>
 </appSettings>
```

**Onde:**

**key** = nome sugestivo da String de Conexão.
**Value** = Dados para a conexão com o banco de dados.
**Database** = nome do banco de dados.
**Data Source** = Onde está armazenado o banco de dados, pode ser o Ip ou o endereço local.
**User id** = nome do usuário credenciado para acessar o banco de dados.
**Password** = senha de acesso do MySql para acessar o Banco de Dados.
**Pooling** = permite trabalhar com vários acessos utilizando a mesma as conexões;

## Script Conpleto: Web.Config com a String de Conexão

```
<?xml version=”1.0”?>
<configuration>
<appSettings>
    <add key=”strConexao” value=”Database = aulabruno; Data Source = localhost; User id = root;
    Password=; pooling = false; “/>
 </appSettings>

 <system.web>
  <compilation debug=”false” targetFramework=”4.0”>
   <assemblies>
            <add assembly=”MySql.Data, Version=6.6.3.0, Culture=neutral,
            PublicKeyToken=C5687FC88969C44D”/>
   </assemblies>
  </compilation>
  <httpRuntime/>
 </system.web>
</configuration>
```

# Classe Mapped

A classe **Mapped** é uma classe de mapeamento.
Ela pode ter qualquer nome.
Esta é a classe possui os métodos de **conexão**, **comando, mapeamento** e **execução** com o banco de dados.

**Métodos Comuns:**

Método de Conexão - Cria o objeto de Conexão
Método de Comandos SQL - Cria o objeto e valida o comando a ser executado
Método Adapter - Executa o comando
Método de Parametrização - Valida as entradas de dados antes de executar o comando Sql.

# Criando a classe Mapped

## Importar as seguintes Bibliotecas

```
//As bibliotecas MySql só aparecerão se a referência ao MyqlData estivar correta
using MySql.Data.MySqlClient;
using System.Configuration;
using System.Data;
```

## Classe Mapped sem comentáiro

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using MySql.Data.MySqlClient;
using System.Configuration;
using System.Data;

public class Mapped{

 //Método para abrir a conexão
 public static IDbConnection Connection(){
   MySqlConnection objConexao = new
   MySqlConnection(ConfigurationManager.AppSettings["strConexao"]);
   objConexao.Open();
   return objConexao;
 }

   // Comandos SQL - Cria o objeto e valida o comando a ser executado
   public static IDbCommand Command(string query, IDbConnection objConexao){
      IDbCommand command = objConexao.CreateCommand();
      command.CommandText = query;
      return command;
    }

    // Funciona como uma ponte entre os dados desconexos e conexos
    public static IDataAdapter Adapter(IDbCommand command){
       IDbDataAdapter adap = new MySqlDataAdapter();
       adap.SelectCommand = command;
       return adap;
    }

   // Parametrização
   // Valida as entradas de dados antes de executar o comando Sql
    public static IDbDataParameter Parameter(string nomeDoParametro, object valor){
        return new MySqlParameter (nomeDoParametro, valor);
    }

}
```

# Comentando os métodos da classe Mapped

## Método de Connection- Comentado

```
// IDbConnection  - Representa uma fonte de conexão com banco de dados
public static IDbConnection Connection() {
            // A linha abaixo cria um objeto de conexão com o MySql
            // Observe que este objeto recebe a string de conexão "strConexao".
            // A string de conexão está dentro do "AppSettings"
            // A AppSettings está dentro do "Configuration" que está o Web.Config
MySqlConnection objConexao = new
MySqlConnection(ConfigurationManager.AppSettings["strConexao"]);
     // O objeto abre a conexão com o banco de dados
      objConexao.Open();
     // Executa e retorna a conexão
      return objConexao;
}
```

---

## Método Command - Comentado:

```
// Comandos SQL - Cria o objeto e valida o comando a ser executado
// IDbCommand - Representa uma instrução SQL que é executado quando conectada a uma fonte de dados, e implementada
pelos provedores de dados.NET Framework que acessam bancos de dados relacionais.

public static IDbCommand Command(string query, IDbConnection objConexao) {
     // O método recebe uma query que será uma sql a ser executada
     // Para executar um comando no Banco de Dados a Conexão deve estar aberta por isso ele recebe o objConexao
      // O objeto command recebe a conexão e cria um comando na conexão que   está sendo executado
       IDbCommand command = objConexao.CreateCommand();
      // O objeto comando.CommandText recebe a query que é uma expressão SQL
       command.CommandText = query;
     //Executa e retorna o comando
      return command;
}
```

---

## Método Adapter - Comentado

```
IDataAdapter  - trabalha como ponte entre dados desconexos e conexos.  Após carregado o DataSet, será possível uma
manipulação desconexa destes dados em memória.
public static IDataAdapter Adapter( IDbCommand command) {
     // O método executa um commando validado, por isso ele recebe um  IDbCommand
       IDbDataAdapter adap = new MySqlDataAdapter();

     // O objeto executa e recebe os dados do Banco de Dados
       adap.SelectCommand = command;
       return adap;
}
```

---

## Método Parameter – Comentado

```
// Valida as entradas de dados antes de executar o comando Sql
public static IDbDataParameter Parameter  (  string nomeDoParametro,  object valor) {
     // Vai validar as entradas de dados e retornar os valores para serem trabalhado no sistema
      return new MySqlParameter(nomeDoParametro, valor);
}
```

## Criando as classes Básicas

```
// Classe TipoEmpresa
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
public class TipoEmpresa {
   private int tip_id;
   private string tip_descricao;
   public string Tip_descricao{
      get { return tip_descricao; }
      set { tip_descricao = value; }
   }
   public int Tip_id {
      get { return tip_id; }
      set { tip_id = value; }
   }
}
```

## Observe que esta classe tem uma chave estrangeira em sua tabela no Banco de Dados

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

public class Empresa{
   private int emp_id;
   private string emp_nome;
   private string emp_rua;
   private string emp_numero;
   private string emp_bairro;
   private string emp_cidade;
   private string emp_estado;

  //Atenção para a chave estrangeira
   private TipoEmpresa tip_id;

   public global::TipoEmpresa Tip_id {
      get { return tip_id; }
      set { tip_id = value; }
   }


   public int Emp_id {
      get { return emp_id; }
      set { emp_id = value; }
   }

   public string Emp_nome  {
      get { return emp_nome; }
      set { emp_nome = value; }
   }

   public string Emp_rua {
      get { return emp_rua; }
      set { emp_rua = value; }
   }

   public string Emp_numero {
      get { return emp_numero; }
      set { emp_numero = value; }
   }

   public string Emp_bairro {
      get { return emp_bairro; }
      set { emp_bairro = value; }
   }

   public string Emp_cidade {
      get { return emp_cidade; }
      set { emp_cidade = value; }
   }

   public string Emp_estado {
      get { return emp_estado; }
      set { emp_estado = value; }
   }

}
```

# Classe de Persistência

Classes de persistências, que são as classes que trabalharão com o acesso ao banco de dados (Select, Delete, Insert, Update).

Nem toda classe precisa obrigatoriamente de uma classe de persistência.

```
// importar a bibliotecas
  using System.Data;
```

## Persistência da classe Tipo Empresa

### Método Insert

```
public static int Insert(TipoEmpresa tipo) {
  int retorno = 0;
  try{
    IDbConnection objConexao;  // Abre a conexao
    IDbCommand objCommand;  // Cria o comando
    string sql = "INSERT INTO tip_tipoempresa (tip_descricao) "+"VALUES (?tip_descricao)";
    objConexao = Mapped.Connection();
    objCommand = Mapped.Command(sql, objConexao);
   objCommand.Parameters.Add(Mapped.Parameter("?tip_descricao", tipo.Tip_descricao));
    objCommand.ExecuteNonQuery();   // utilizado quando cdigo não tem retorno, como seria o caso do SELECT
    objConexao.Close();
    objCommand.Dispose();
    objConexao.Dispose();
 }catch (Exception e){
     retorno = -2;
 }
  return retorno;
}
```

---

---

### Método Insert Comentado

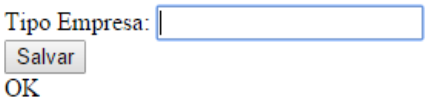
```
// Insere todos os dados do objeto (Classe TipoEmpresa) no BD é muito mais
prático passar um objeto todo de uma vez que passar atributo por atributo

public static int Insert(TipoEmpresa tipo){
    // 0 = caso de Sucesso
    // -2 = caso de falha de conexão
    int retorno = 0;

    try{
     IDbConnection objConexao; // Cria obj conexao
     IDbCommand objCommand; // Cria obj comando

     //SINTAXE: "INSERT INTO nomeDaTabela (nomeDasColunas) VALUE
    (?parametrização)";
     string sql = "INSERT INTO tipoempresa (tip_descricao) " +
```

```
      "VALUES (?tip_descricao)";

      objConexao = Mapped.Connection();  // Abre a conexão

       //atribiu ao objeto commando a Sql e o objeto de conexão
       objCommand = Mapped.Command(sql, objConexao);

      // Parametriza os dados - Adiciona os dados para parametrização
        objCommand.Parameters.Add(Mapped.Parameter("?tip_descricao",
        tipo.Tip_descricao));

        // Executa o comando
        // Comando ExecuteNoQuery - utilizado quando codigo não tem retorno,
        como seria o caso de SELECT
        objCommand.ExecuteNonQuery();

        objConexao.Close();  //Fecha a conexão

        //Disponibiliza o objeto conexão e o objeto comando para serem
        utilizados novametne
           objCommand.Dispose();
           objConexao.Dispose();
         }catch (Exception e){
              retorno = -2;
         }
          return retorno;
    }

      ----------------------------------------------------------------------------------------------------
      ----------------------------------------------------------------------------------------------------
```

## Testando - Insert

### Código de Inserção ASPX

```
<asp:Label ID="lblTipoEmpresa" runat="server" Text="Tipo Empresa: "></asp:Label>
<asp:TextBox ID="txbTipoEmpresa" runat="server"></asp:TextBox>
<br />
<asp:Button ID="btnSalvar" runat="server" Text="Salvar" OnClick="btnSalvar_Click" />
<br />
<asp:Label ID="lblConfirmacao" runat="server" Text=""></asp:Label>
```




### Código de Inserção ASPX.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class Paginas_Insert1 : System.Web.UI.Page{
  protected void Page_Load(object sender, EventArgs e){  }

  protected void btnSalvar_Click(object sender, EventArgs e){

    TipoEmpresa tipo = new TipoEmpresa();
    tipo.Tip_descricao = txbTipoEmpresa.Text;
```

```
        switch(TipoEmpresaDB.Insert(tipo)){
            case 0:
                lblConfirmacao.Text = "OK";
                break;
            case -2:
                lblConfirmacao.Text = "ZICOU";
                break;
        }
    }
}
```

---

---

## Método Insert da Tabela Empresa

Deve-se observar que na tabela Empresa temos uma chave estrangeira, esta é a única diferença que encontramos em relação aos outros métodos da persistência.

Os demais métodos seguem o mesmo padrão da Classe de Persistência TipoEmpresaDB.

```
public static int Insert(Empresa empresa){
    int errNumber = 0;
    try{
        IDbConnection objConexao;
        IDbCommand objCommand;
        string sql = "INSERT INTO empresa(";
        sql += "emp_nome,";
        sql += "emp_rua,";
        sql += "emp_numero,";
        sql += "emp_bairro,";
        sql += "emp_cidade,";
        sql += "emp_estado,";
        sql += "tip_id";
        sql += ")";
        sql += " VALUES (";
        sql += "?emp_nome,";
        sql += "?emp_rua,";
        sql += "?emp_numero,";
        sql += "?emp_bairro,";
        sql += "?emp_cidade,";
        sql += "?emp_estado,";
        sql += "?tip_id";
        sql += ")";
        objConexao = Mapped.Connection();
        objCommand = Mapped.Command(sql, objConexao);
            objCommand.Parameters.Add(Mapped.Parameter("?emp_nome", empresa.emp_nome));
            objCommand.Parameters.Add(Mapped.Parameter("?emp_rua", empresa.emp_rua));
            objCommand.Parameters.Add(Mapped.Parameter("?emp_numero",
            empresa.emp_numero));
            objCommand.Parameters.Add(Mapped.Parameter("?emp_bairro", empresa.emp_bairro));
            objCommand.Parameters.Add(Mapped.Parameter("?emp_cidade", empresa.emp_cidade));
            objCommand.Parameters.Add(Mapped.Parameter("?emp_estado", empresa.emp_estado));
            objCommand.Parameters.Add(Mapped.Parameter("?tip_id", empresa.Tip_id.Tip_id));
            // Chave Extrangeira

        objCommand.ExecuteNonQuery();
        objConexao.Close();
        objCommand.Dispose();
```

```
            objConexao.Dispose();
        }catch (Exception ex){
            errNumber = -2;
        }
        return errNumber;
    }
}
```

---

---

## Persistência – SELECT * From...

```
Public static DataSet SelectAll() {

    DataSet ds = new DataSet();
    IDbConnection objConnection;
    IDbCommand objCommand;
    IDataAdapter objDataAdapter;
    objConnection = Mapped.Connection();
     objCommand = Mapped.Command("SELECT * FROM tipoempresa ORDER BY tip_descricao",
     objConnection);
    objDataAdapter = Mapped.Adapter(objCommand);

    objDataAdapter.Fill(ds); // O objeto DataAdapter vai preencher o
     DataSet com os dados do BD, O método Fill é o responsável por   preencher o DataSet
    objConnection.Close();
    objCommand.Dispose();
    objConnection.Dispose();
    return ds;
  }
}
```

---

---

## Persistência – SELECT * From... Comentado

```
// O DataSet é um objeto que armazena grande quantidade de dados
Public static DataSet SelectAll() {
// Aqui não é necessário criar um try/catch, pois se um erro ocorrer com a conexão, o dataset ficará
vazio.  Se o dataset ficar vazio ou não temos dados no BD ou o código está errado

    DataSet ds = new DataSet();
    IDbConnection objConnection;
    IDbCommand objCommand;
    IDataAdapter objDataAdapter;    //  trabalha como ponte entre dados desconexos e conexos. Após
            carregado o DataSet, será possível uma manipulação desconexa destes dados em memória.

    objConnection = Mapped.Connection();
```

```
        objCommand = Mapped.Command("SELECT * FROM tipoempresa ORDER BY tip_descricao",
       objConnection);
        objDataAdapter = Mapped.Adapter(objCommand);

        // O objeto DataAdapter vai preencher o DataSet com os dados do BD, O método Fill é o
        responsável por preenchre o DataSet
        objDataAdapter.Fill(ds);
        objConnection.Close();
        objCommand.Dispose();
        objConnection.Dispose();
        return ds;
    }
}
```

---

---

## Carregar DropDownList

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data;

public partial class Paginas_CarregarDropDown : System.Web.UI.Page{

    protected void Page_Load(object sender, EventArgs e){
        if (!IsPostBack){
         //Carregar um DropDownList com o Banco de Dados
         DataSet ds = TipoEmpresaDB.SelectAll();
         ddl.DataSource = ds;
         ddl.DataTextField = "tip_descricao"; // Nome da coluna do Banco de dados
         ddl.DataValueField = "tip_id"; // ID da coluna do Banco
         ddl.DataBind();
         ddl.Items.Insert(0, "Selecione");
        }

    }
```

---

---

## Carregar GridView

```
public partial class Paginas_CarregarGrid : System.Web.UI.Page {

    protected void Page_Load(object sender, EventArgs e) {
        if (!IsPostBack) {
            CarregarGrid();
        }
    }

    public void CarregarGrid() {
        //DataSet ds = new DataSet();
        DataSet ds = TipoEmpresaDB.SelectAll();
        int qtd = ds.Tables[0].Rows.Count;
        if (qtd > 0) {
            gdv.DataSource = ds.Tables[0].DefaultView;
            gdv.DataBind();
```

```
            gdv.Visible = true;
            lbl.Text = "Foram encontrados " + qtd + " de registros";
        } else {
            gdv.Visible = false;
            lbl.Text = "Não foram encontrado registros...";
        }
    }
}
```

---

---