Protótipo de um ORM para a plataforma .NET

Orientando: Thiago Boufleuhr

Orientador: Adilson Vahldick

Sequência de Apresentação

i. Introdução
ii. Objetivos
iii. Fundamentação teórica
iv. Desenvolvimento do protótipo
v. Estudo de caso
vi. Conclusões
vii. Extensões

Introdução

- Automatização de processos repetitivos
- Necessidade de um *framework*
- Problema da tenacidade dos dados
- Persistência na plataforma .NET
- Mapeamento objeto relacional

Objetivos

- Permitir o mapeamento de objetos C# para as entidades relacionais através do uso de atributos customizados;
- Simplificar o uso de chaves primárias compostas;
- Permitir o uso da LINQ para realizar consultas ao banco de dados;
- Eliminar a necessidade de mapeamento através de arquivos *eXtensible Markup Language* (XML);
- Simplificar a utilização de relacionamentos entre as classes mapeadas.

Fundamentação Teórica

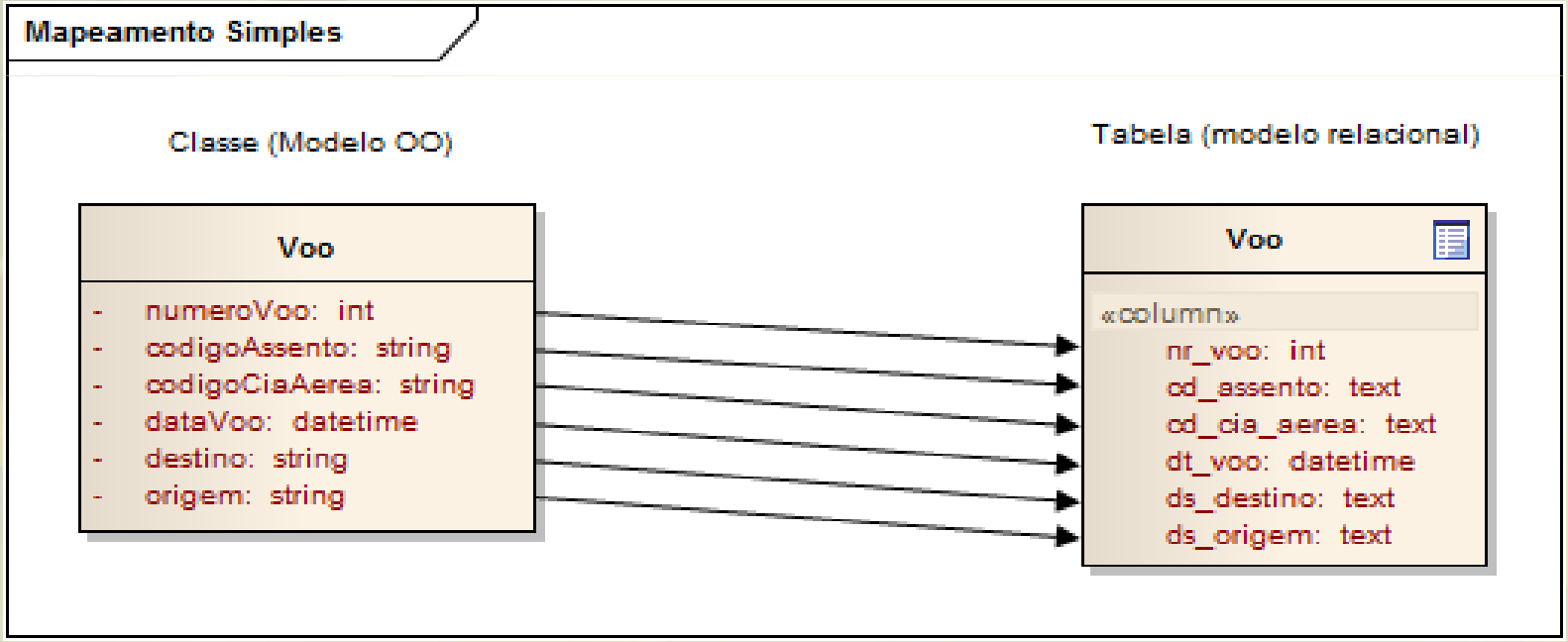
i. Mapeamento objeto relacional
ii. Mapeamento de herança
iii. Mapeamento de relacionamentos
iv. Tecnologia LINQ
v. Atributos customizados

Mapeamento Objeto Relacional

- Conceito básico de mapeamento
- Vantagens da utilização de mapeamento
- Formas de mapear uma classe para uma entidade
- Transformação do modelo OO para o modelo relacional

Mapeamento Básico

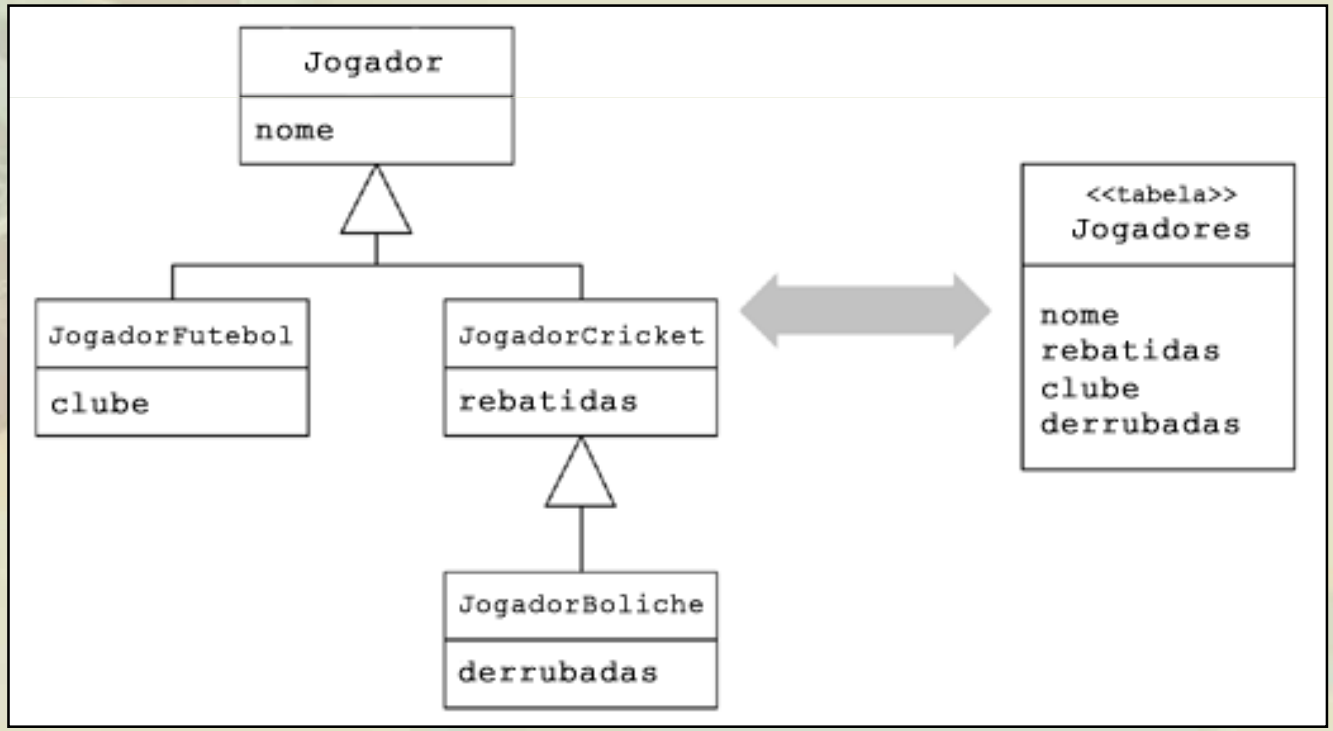
- Consiste em mapear um atributo da classe para uma coluna da tabela do banco de dados



Fonte: Adaptador de Fowler (2002)

Mapeamento de Herança

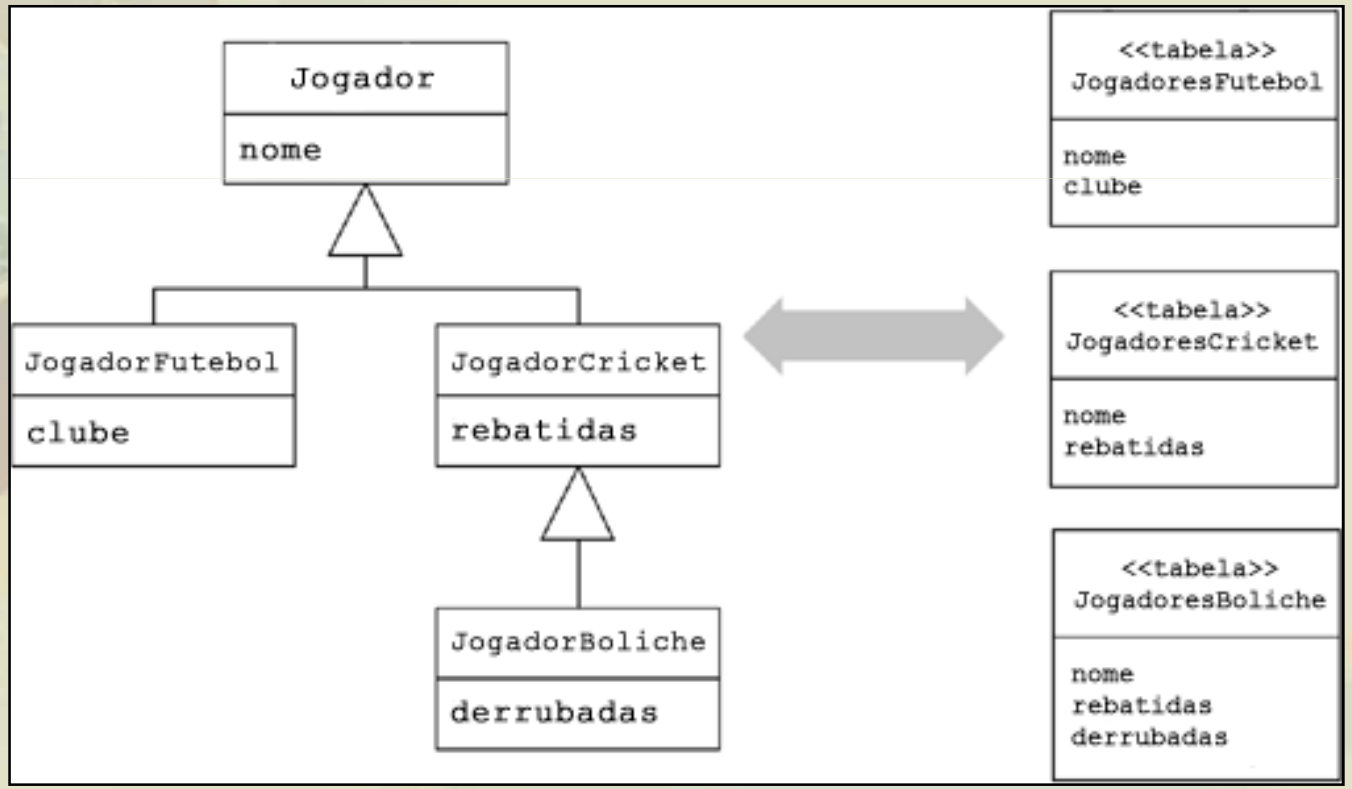
- Uma tabela para todas as classes da hierarquia



Fonte: Adaptado de Fowler (2002).

Mapeamento de Herança

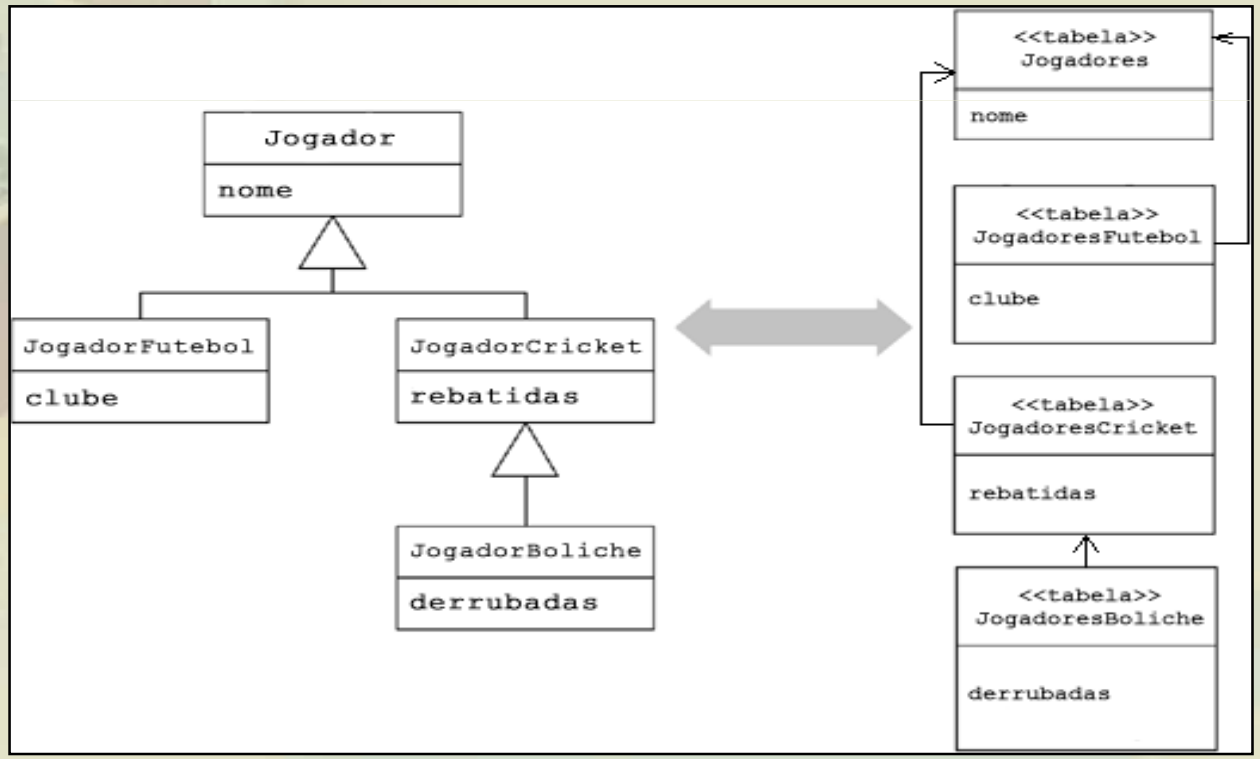
- Uma tabela para cada classe concreta



Fonte: Adaptado de Fowler (2002).

Mapeamento de Herança

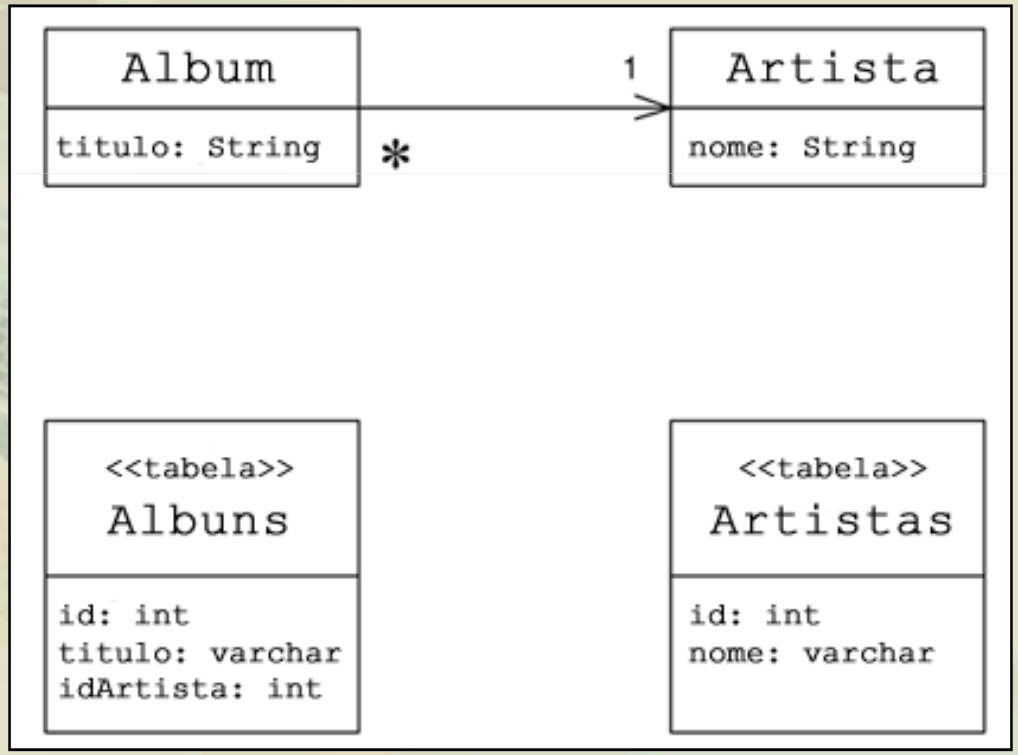
- Uma tabela por classe (abstrata e concreta)



Fonte: Adaptado de Fowler (2002).

Relacionamentos

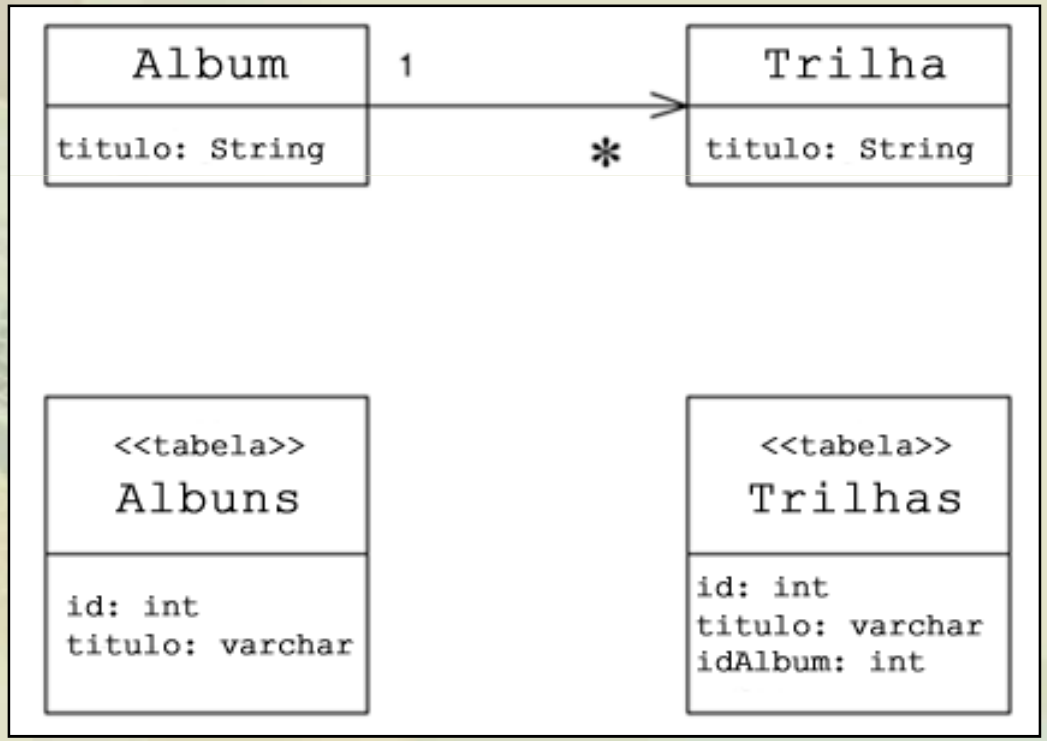
- Relacionamento um-para-um



Fonte: Adaptado de Fowler (2002).

Relacionamentos

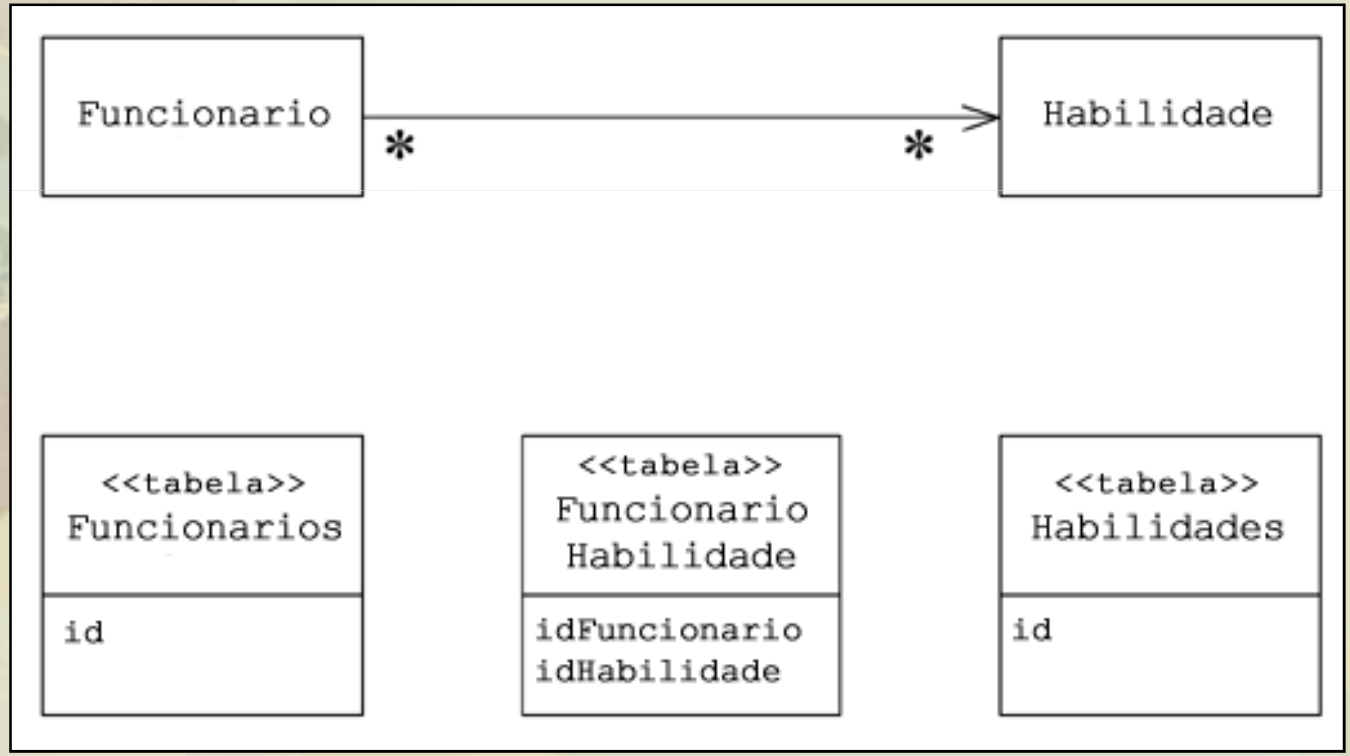
- Relacionamento um-para-muitos



Fonte: Adaptado de Fowler (2002).

Relacionamentos

- Relacionamento muitos-para-muitos



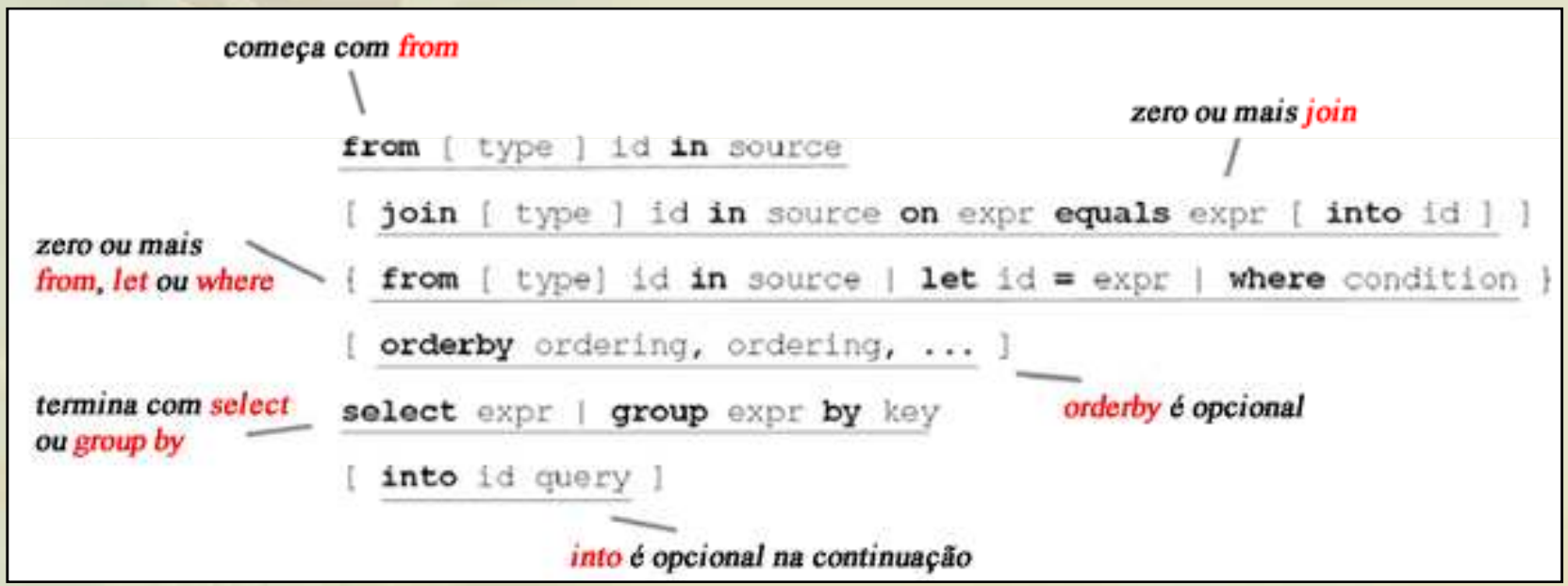
Fonte: Adaptado de Fowler (2002).

Tecnologia LINQ

- O que é LINQ ?
- Objetivos da linguagem LINQ
- Cenários de utilização
- Vantagens

Tecnologia LINQ

- Sintaxe da linguagem:



Fonte: Adaptado de Marguerie; Eichert; Wolley (2009, p. 99).

Tecnologia LINQ

- Exemplos:

```
from customer in customers
where customer.Name.StartsWith("A") && customer.Orders.Count > 0
orderby customer.Name
select new { customer.Name, customer.Orders }
```

Fonte: Marguerie; Eichert; Wolley (2009, p. 12).

```
var processes =
  Process.GetProcesses()
    .Where(process => process.WorkingSet64 > 20*1024*1024)
    .OrderByDescending(process => process.WorkingSet64)
    .Select(process => new { process.Id,
                             Name=process.ProcessName });
```

Fonte: Adaptado de Marguerie; Eichert; Wolley (2009, p. 85).

Atributos Customizados

- O que são ?
- Construção de um atributo customizado
- Formas de utilização

Atributos Customizados

- Exemplo de construção de um atributo customizado:

```
// Um Atributo Customizado
public sealed class DescricaoVeiculoAttribute : System.Attribute
{
    private String descricao;

    public DescricaoVeiculoAttribute()
    {

    }

    public DescricaoVeiculoAttribute(String descricao)
    {
        this.descricao = descricao;
    }

    public String Descricao
    {
        get { return this.descricao; }
        set { this.descricao = value; }
    }
}
```

Fonte: Adaptado de Troelsen (2007, p. 547).

Atributos Customizados

- Exemplo de utilização:

```
// Adicionando uma descrição através de "propriedades nomeadas"
[Serializable]
[DescricaoVeiculo(Descricao = "Minha adorada Harley")]
public class Motocicleta
{

}

[Serializable]
[Obsolete("Utilize outro veículo!")]
[DescricaoVeiculo(Descricao = "Conduzida pela velha égua cinzenta")]
public class Carroca
{

}

[DescricaoVeiculo(Descricao = "Um carro longo, lento, mais com muitas qualidades")]
public class MotorHome
{

}
```

Fonte: Adaptado de Troelsen (2007, p. 547).

Desenvolvimento do Protótipo

i. Requisitos
ii. Principais casos de uso
iii. Diagramas de atividades
iv. Ferramentas utilizadas
v. Técnicas utilizadas na implementação

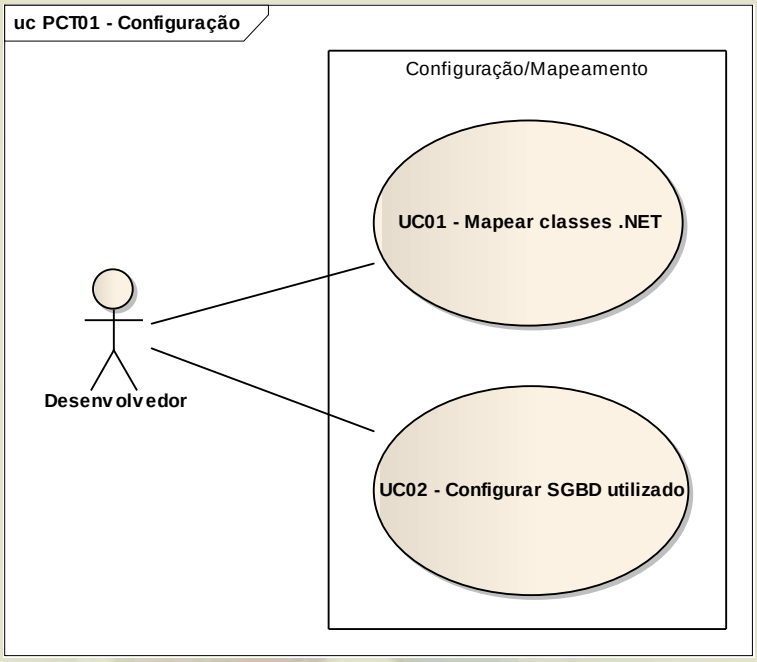
Requisitos

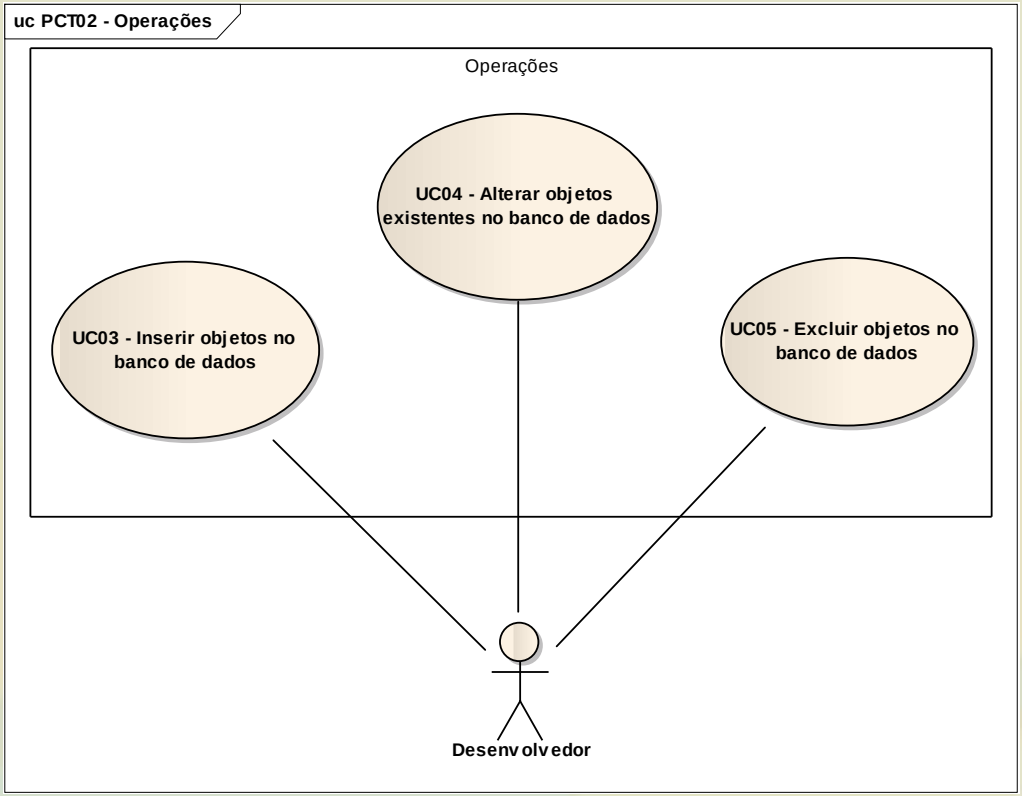
- Principais Requisitos Funcionais:
  - O framework deverá permitir o mapeamento de classes C# para o modelo relacional utilizando atributos customizados.
  - O framework deverá permitir que seja informado em tempo de execução o SGBD que será utilizado.
  - O framework deverá suportar a operação de inserção de objetos mapeados em um banco de dados.
  - O framework deverá suportar a operação de alteração de objetos mapeados existentes em um banco de dados.
  - O framework deverá suportar a operação de exclusão de registros mapeados em um banco de dados.
  - O framework deverá disponibilizar o uso customizado do LINQ para consultas ao banco de dados através de classes mapeadas.

Requisitos

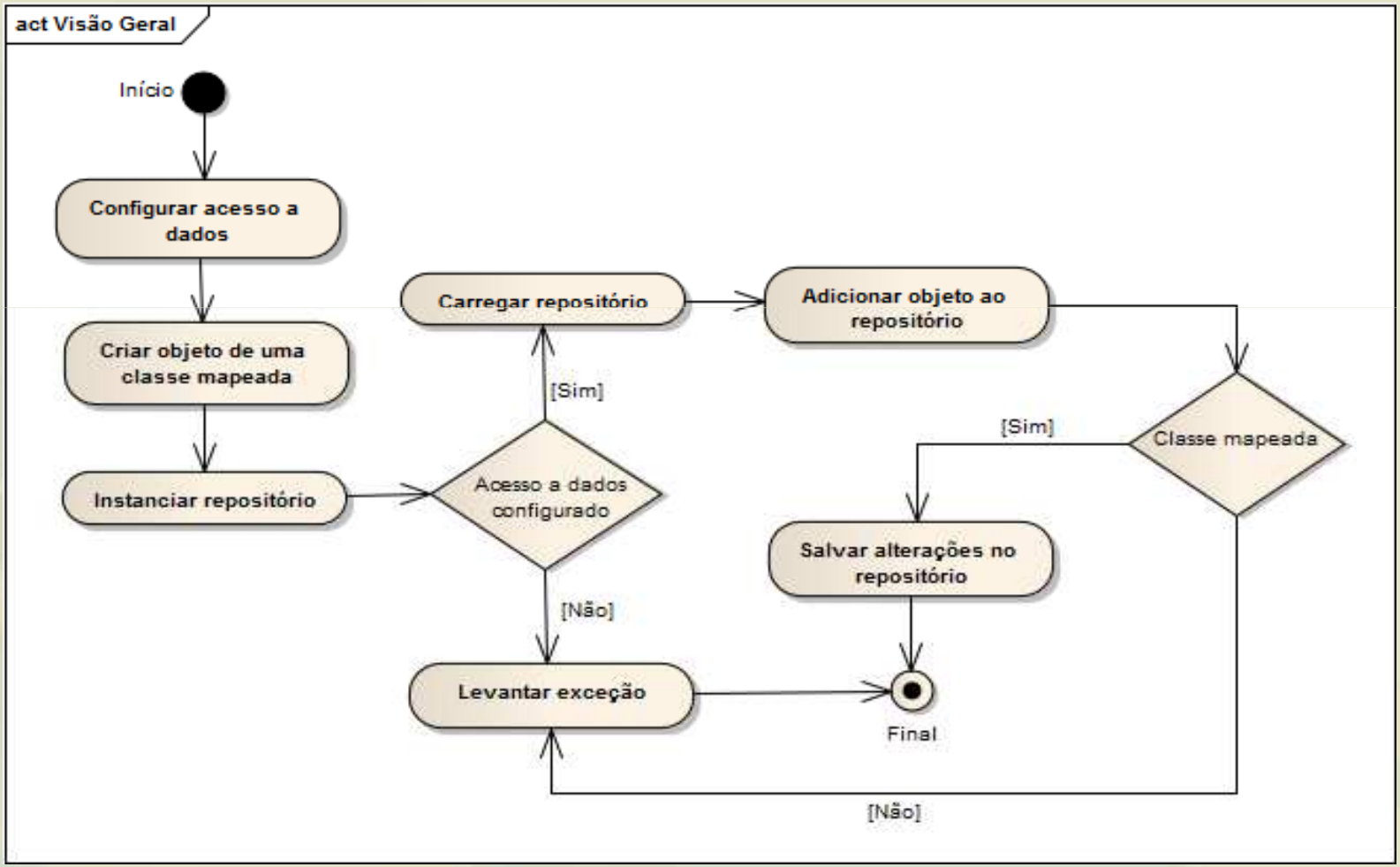
- Principais Requisitos Não Funcionais:
  - O framework deverá suportar os SGBD's SQLServer, MySQL e PostgreSQL.
  - O framework deverá ser desenvolvido utilizando a linguagem C# juntamente com o .NET Framework 3.5 ou superior.
  - O framework deverá manter um *cache* dos metadados das classes.
  - O framework deverá gerar um *log* em memória dos últimos 10 comandos enviados ao SGBD.

Principais Casos de Uso

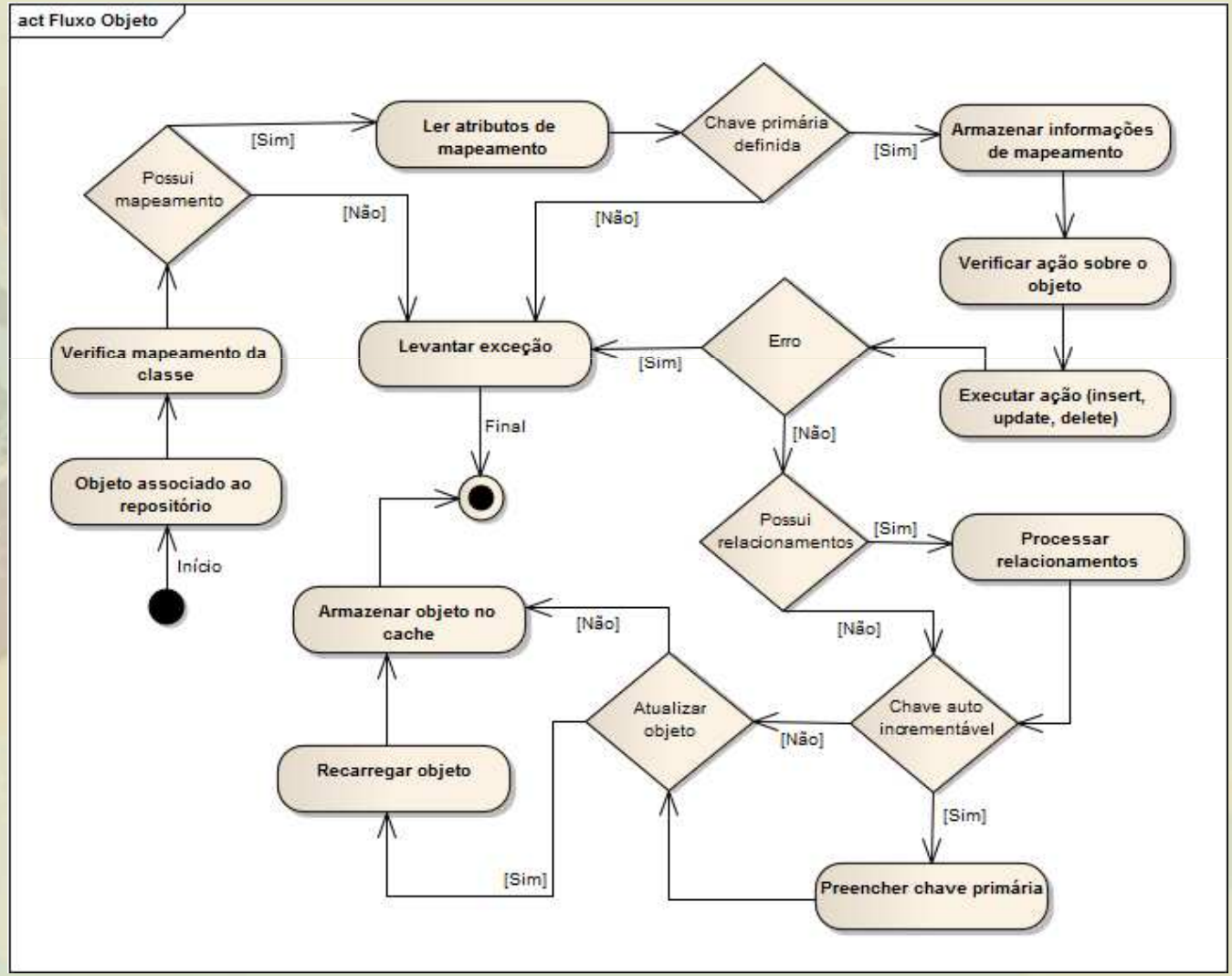




Diagramas de Atividades



Diagramas de Atividades



Ferramentas Utilizadas

- Microsoft .NET Framework 3.5 SP1
- Linguagem C# (*Csharp*)
- Microsoft Visual Studio 2008 Professional
- SQL Server 2008
- MySQL Server 5.1
- PostgreSQL 8.4.2

Técnicas Utilizadas

- Metadados
- Reflexão computacional
- Carregamento tardio (*lazy loading*)
- Repositório de dados (*Repository Pattern*)

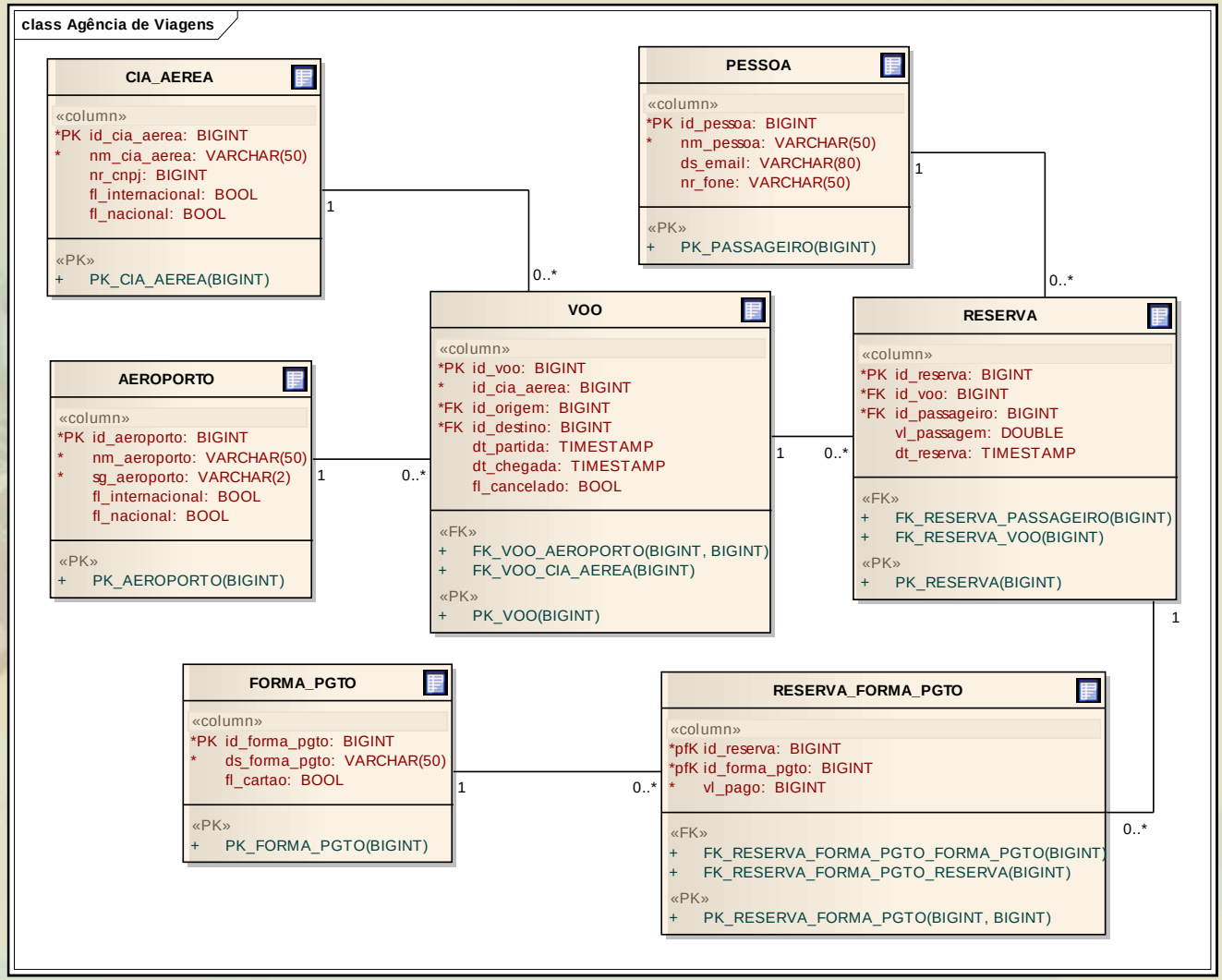
Estudo de Caso

i. Objetivo
ii. Modelo entidade relacionamento
iii. Modelo de classes
iv. Principais classes mapeadas

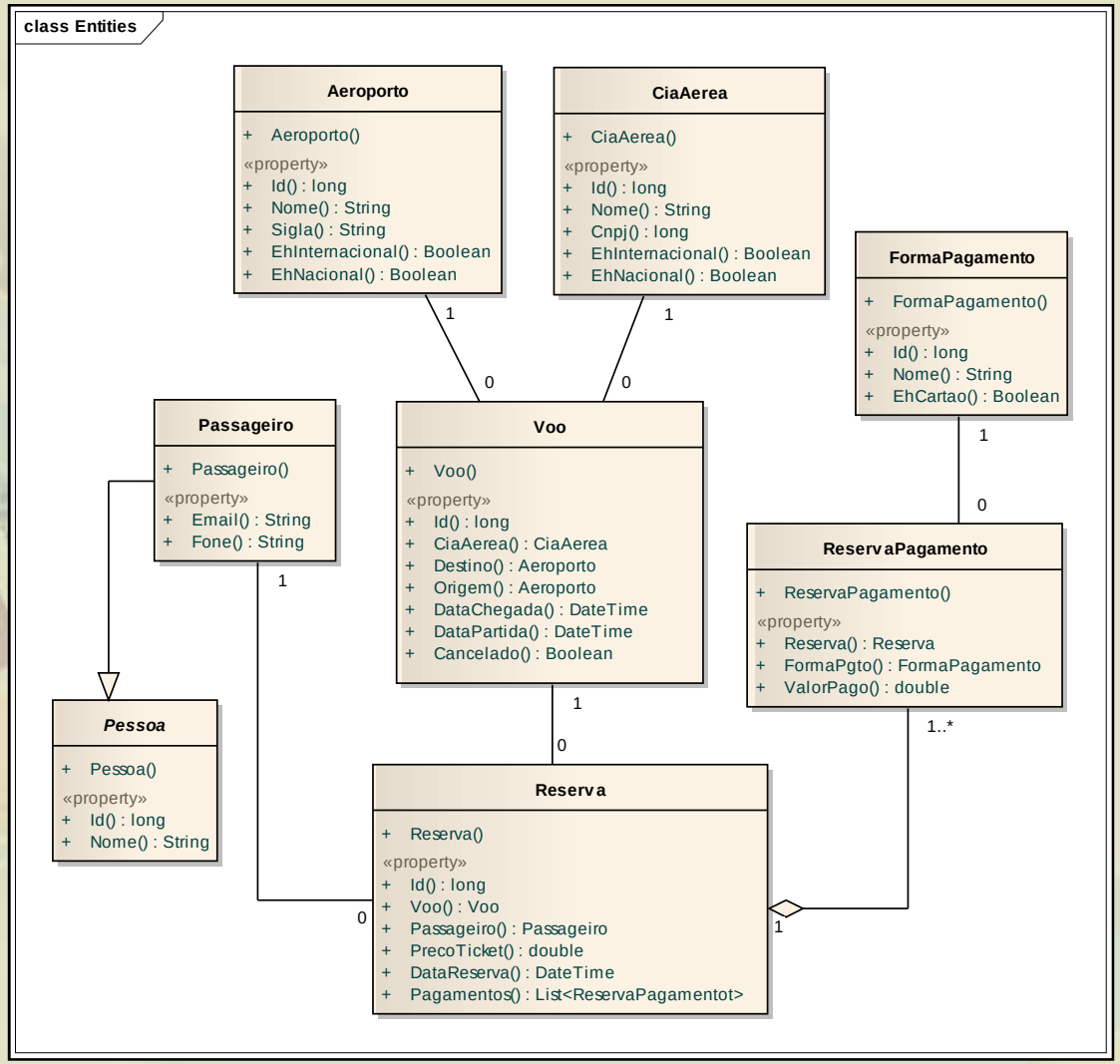
Objetivo

- Aplicar os recursos desenvolvidos no protótipo utilizando um aplicativo exemplo de simulação de uma Agência de Turismo;

Entidade Relacionamento



Modelo de Classes



Principais Classes Mapeadas

```
[TableMap]
public abstract class Pessoa
{
    [ColumnMap(ColumnName="ID_PESSOA", IsAutoIncrement=true, IsPrimaryKey=true, AllowNull=false)]
    public long Id { get; set; }

    [ColumnMap(ColumnName="NM_PESSOA", AllowNull=false)]
    public String Nome { get; set; }

    public Pessoa()
    {
    }
}

[TableMap(TableName = "PESSOA")]
public class Passageiro : Pessoa
{
    [ColumnMap(ColumnName = "DS_EMAIL")]
    public String Email { get; set; }

    [ColumnMap(ColumnName = "NR_FONE")]
    public String Fone { get; set; }

    public Passageiro()
    {
    }

    public override string ToString()
    {
        return this.Nome;
    }
}
```

Principais Classes Mapeadas

```
[TableMap]
public class Voo
{
    [ColumnMap(ColumnName = "ID_VOO", AllowNull = false, IsAutoIncrement = true, IsPrimaryKey = true)]
    public long Id { get; set; }

    [ColumnMap(ColumnName = "ID_CIA_AEREA", AllowNull = false)]
    public CiaAerea CompanhiaAerea { get; set; }

    [ColumnMap(ColumnName = "ID_DESTINO", AllowNull = false)]
    public Aeroporto Destino { get; set; }

    [ColumnMap(ColumnName = "ID_ORIGEM", AllowNull = false)]
    public Aeroporto Origem { get; set; }

    [ColumnMap(ColumnName = "DT_CHEGADA")]
    public DateTime DataChegada { get; set; }

    [ColumnMap(ColumnName = "DT_PARTIDA")]
    public DateTime DataPartida { get; set; }

    [ColumnMap(ColumnName = "FL_CANCELADO")]
    public Boolean Cancelado { get; set; }

    public Voo()
    {
    }

    public override string ToString()
    {
        return this.Id.ToString();
    }
}
```

Principais Classes Mapeadas

```
[TableMap(TableName="RESERVA_FORMA_PGTO")]
public class ReservaPagamento
{
    [ColumnMap(ColumnName="ID_RESERVA", AllowNull = false, IsPrimaryKey = true)]
    public Reserva Reserva { get; set; }

    [ColumnMap(ColumnName = "ID_FORMA_PGTO", AllowNull = false, IsPrimaryKey = true)]
    public FormaPagamento Payment { get; set; }

    [ColumnMap(ColumnName = "VL_PAGO", AllowNull = false)]
    public double ValorPago { get; set; }

    public ReservaPagamento()
    {

    }
}
```

Principais Classes Mapeadas

```
[TableMap]
public class Reserva
{
    [ColumnMap(ColumnName="ID_RESERVA", AllowNull=false, IsAutoIncrement=true, IsPrimaryKey=true)]
    public long Id { get; set; }

    [ColumnMap(ColumnName = "ID_VOO", AllowNull = false)]
    public Voo Voo { get; set; }

    [ColumnMap(ColumnName = "ID_PASSAGEIRO", AllowNull = false)]
    public Passageiro Passageiro { get; set; }

    [ColumnMap(ColumnName = "VL_PASSAGEM")]
    public double PrecoTicket { get; set; }

    [ColumnMap(ColumnName = "DT_RESERVA")]
    public DateTime DataReserva { get; set; }

    [ColumnMap(Relationship = RelationshipType.ManyToMany)]
    public List<ReservaPagamento> Pagamentos { get; set; }

    public Reserva()
    {

    }
}
```

Conclusões

- Necessidade de um *framework* ORM
- Carência de um ORM na plataforma .NET
- Padrões Microsoft de desenvolvimento
- Foco na simplicidade de uso
- Não manipulação de arquivos externos

Extensões

- Aprimoramento do sistema de *cache*
- Integração com demais SGBD's
- Geração DDL (Data Definition Language)
- Utilizar *lazy load* através do padrão *proxy*
- Suportar POCO (*plain old CLR objects*)
- Acrescentar suporte a *multi-threading*
- Transferência remota de objetos

Obrigado !