UNIVERSIDADE DE CUIABÁ

FACULDADE DE SISTEMAS DE INFORMAÇÃO

FERNANDO VICTOR PEREIRA SANTIAGO

VANTAGENS DE UTILIZAR OBJECT-RELATIONAL MAPPING (ORM) EM .NET

Cuiabá

2015

FERNANDO VICTOR PEREIRA SANTIAGO

VANTAGENS DE UTILIZAR OBJECT-RELATIONAL MAPPING (ORM) EM .NET

Trabalho de Curso submetido à Universidade de Cuiabá como parte dos requisitos necessários para a obtenção do Grau de Bacharel em Sistemas de Informação.

Cuiabá
2015

Dedico este trabalho a minha mãe que sempre acreditou em mim e me mostrou o caminho da vitória com seus exemplos e para todos que contribuíram direta ou indiretamente em minha formação acadêmica.

AGRADECIMENTOS

Agradeço a todos que contribuíram no decorrer desta jornada.

Santiago, Fernando Victor Pereira Santiago. **Vantagens de Utilizar Object-Relational Mapping(ORM) em .NET**. 2015. 55. Trabalho de Conclusão de Curso, Universidade de Cuiabá, Cuiabá, 2015.

RESUMO

Com a popularização dos bancos de dados relacionais, a maioria dos softwares desenvolvidos atualmente utiliza banco de dados relacionais visando organização, múltiplos acessos, flexibilidade, integridade da informação e melhor gestão da informação. Os softwares desenvolvidos utilizam vários tipos de banco de dados(MySQL, Oracle, SQL Serve, PostgreSQL e etc) cada qual utiliza uma API especifica para fazer a conexão com banco de dados e realizar transações. Em .NET utiliza-se ADO.net para fornecer acesso a dados relacionais que podem ser acessados por três objetos diferentes OLE DB, SQL e ODBC.

A vários problemas ao utilizar ADO.net, se ao longo do tempo ver se a necessidade de trocar de banco dados também teria que ser trocado todo o código de configuração, conexão e até scripts que foram feitos pois existe diferente sintaxe entre os bancos. A manutenção de serviços de acesso de dados e muito complexa visto que as regras de negócio podem estar mudando diariamente, surgindo a necessidade de inserir, remover e atualizar novos campos. Com essas atualizações seria necessário verificar se todos os métodos que utilizam essas tabelas alteradas estão funcionando corretamente.

Palavras-chave: Net. ORM. NHibernate. SQL Server.

Santiago, Fernando Victor Pereira Santiago. **Advantages of using Object-Relational Mapping(ORM) in .NET**. 2015. 55. Completion of course work, University of Cuiabá, Cuiabá, 2015.

ABSTRACT

With the popularity of relational databases, most software currently developed uses relational database aimed at organization, multiple access, flexibility, Information integrity and better management of information. The software developed using different types of database (MySQL, Oracle, SQL Serve, PostgreSQL e etc) each use a specific API to connect to the database and perform transactions. Technology microsoft .NET is used ADO.net to provide access to relational data that can be accessed by three different objects OLE DB, SQL e ODBC.

The various problems when using ADO.net, over time to see the need to exchange database data also have to be exchanged all the configuration code, connection and even scripts that have been made for there different syntax between banks. Maintaining data access services and complex as the business rules may be changing daily, resulting in the need to insert, remove and update new fields. With these upgrades would be necessary to check that all methods that use these tables changed are working properly.

Palavras-chave: Net. ORM. NHibernate. SQL Server.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1. INTRODUÇÃO

Um dos maiores problemas de desenvolvimento de software esta na produtividade as regras de negocio mudam o tempo todo. Os clientes finais cada vez mais estão exigindo um software mais rápido, seguro e flexível. A criação de scripts para consultas no banco de dados geram muito tempo e esforço, pois se a regra de negocio mudar e o sistema não possuírem testes unitários teria que ver se todas as funcionalidades estão funcionando corretamente.

Uma das maiores vantagem de usar ORM é a independência de banco de dado. Não há necessidade de escrever código específico para um banco de dados particular. Basta iniciar um projeto usando SQL Server e, mais tarde, mudar para MySQL ou PostgreSQL. Alterando algumas linhas de código na configuração do banco de faz com que funcione com outro banco de dados. O armazenamento em memoria também e fator importantíssimo que pesa a favor do ORM, ele guarda o objeto em cash de memoria reduzindo a carga no banco. ORM fornece uma abstração que permite que os desenvolvedores se concentrem na sua lógica de negócio, ao invés de scripts SQL de banco de dados complexos, resultando em uma enorme redução de código e aumento da eficiência do desenvolvedor.

2. OBJETIVO GERAL

O objetivo principal e expor as vantagens ao utilizar ORM, para esse estudo será utilizado ASP.Net c# com .NET Framework 4.5, Nhibernate, SQL Server e PostgreSQL.

2.1. OBJETIVOS ESPECIFICOS

- Desenvolver referencial teórico sobre ADO.Net e ORM.
- Desenvolver dois sistemas com ADO .Net e ORM.
    - Comparar desempenho de métodos CRUD entre os sistemas.
    - Quantificar quantidade de linhas gastas.
    - Realizar alteração complexa nos dois sistemas.

3. ADO.NET – FUNDAMENTOS

ADO.NET é uma família de tecnologias que permite que os desenvolvedores .NET possam interagir com padrão de dados, formas estruturadas e principalmente dados desconectados.

Os aplicativos escritos usando o Framework .NET depende de bibliotecas de classe .NET, que existem em DLL especiais que encapsulam funcionalidades de programação comum em um acesso de fácil formato. A maioria das bibliotecas fornecidos com o Framework .NET aparecem dentro do System namespace. System.IO, por exemplo, incluir classes que permitem interagir com o disco padrão arquivos e fluxos de dados relacionados. A biblioteca System.Security fornece acesso a recursos de criptografia de dados, dentre outras coisas. ADO.NET, expressa através do namespace System.Data, implementa um pequeno conjunto de bibliotecas que faz consumir e manipulação de grandes quantidades de dados de forma simples e direta.

O ADO.NET e bem familiar ao Microsoft SQL Server, há muitos termos familiares Tabelas, linhas, colunas, relações, views; estes Conceitos de ADO.NET são vagamente baseados em suas contrapartes de bancos de dados relacionais. Apesar destas semelhanças, ADO.NET não é uma base de dados relacional, porque ele não inclui a chave "Álgebra Relacional" normalmente encontrados em sistemas de banco de dados robustos. Ele também não tem muitos dos comuns recursos de suporte de tais bancos de dados, incluindo índices, procedimentos armazenados e triggers. Ainda assim, se você se limitar a base criar, ler, atualizar e excluir (CRUD), ADO.NET pode agir como um poderoso banco de dados ainda na memória.

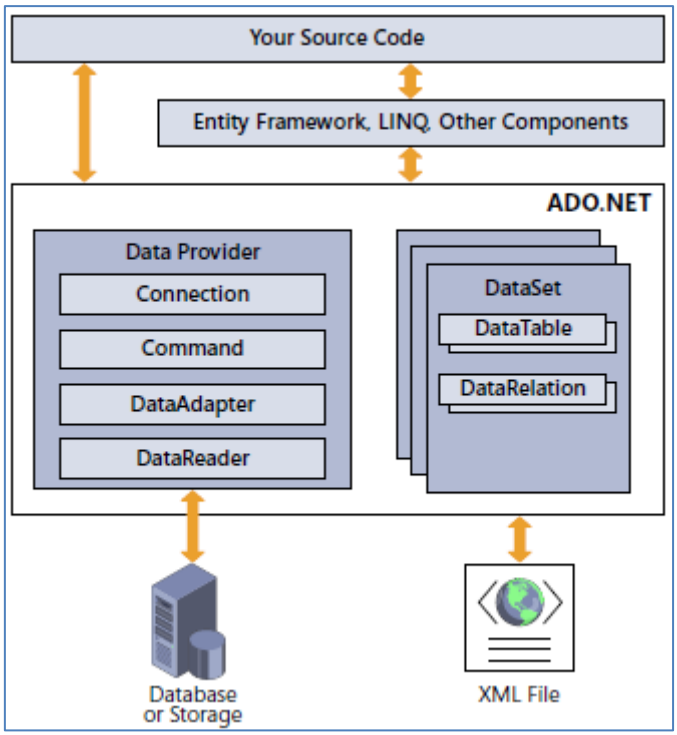
Ao se comunicar com armazenamentos de dados externos, ADO.NET apresenta experiência de dados desconectado. Em plataformas de dados anteriores, incluindo ADO, normalmente faria estabelecendo uma conexão persistente com um banco de dados e utilizar várias formas de bloqueio de registro para gerenciar atualizações de dados seguros e precisos. Mas então veio a Internet. A manutenção de uma conexão de dados de longa data através de rajadas de Conteúdo de texto HTTP já não era uma expectativa realista. A

preferência de ADO.NET direção on-again, off-again conexões de banco de dados reflete essa realidade. Embora esta mudança no paradigma apresentado com ela dificuldades para os desenvolvedores de aplicativos cliente-servidor tradicionais, ele também ajudou a inaugurar a era da enorme escalabilidade e n-tier desenvolvimento que agora é comum a ambos área de trabalho e sistemas web-based.

3.1. PRINCIPAIS COMPONENTES DO ADO.NET

O namespace System.Data inclui muitas classes ADO.NET distintas que trabalham juntos para proporcionar o acesso a dados tabulares. A biblioteca inclui dois grandes grupos de classes: as que gerenciam os dados reais dentro do software e aquelas que se comunicam com dados externos dos sistemas. A Figura 1 mostra as principais partes que compõem uma instância ADO.NET.

Figura 1 – Elementos chaves do ADO Net.



Fonte: Chiavenato (1994, p. 170)

No centro da biblioteca do ADO.Net está o DataTable. Para efeitos semelhantes em tabelas de um banco de dados, o DataTable gerencia todos os valores de dados reais do código-fonte. Cada DataTable contém 0 ou N número de linhas de dados, os valores de dados de cada linha identificada por definições de coluna da tabela.

- Existe uma entrada DataRow para cada registro de dados armazenados dentro de uma tabela, fornecendo acesso para os valores de dados em colunas distintas. ADO.NET inclui métodos que permitem adicionar, apagar, modificar e consultar as linhas de cada objeto DataTable. Para tabelas ligadas para uma área de armazenamento de dados externo todas as mudanças feitas podem ser propagadas de volta para a fonte.
- Opcionalmente, é possível estabelecer ligações entre as tabelas de dados usando entradas DataRelation.
- Instâncias DataView fornecem uma visão limitada ou alterações das linhas em um DataTable.
- As tabelas podem ser agrupadas em um DataSet. Algumas ferramentas que interagem com o ADO.NET exigem que todas as tabelas devem estar ligadas dentro de um DataSet, mas pode se trabalhar com apenas uma única tabela.

Instâncias de DataTable e seus objetos associados são suficientes para trabalhar com dados. Para se conectar a dados externos a partir de um banco de dados, ADO.NET apresenta vários provedores de dados, incluindo um provedor personalizado para o Microsoft SQL Server (oque vai ser usado para exemplificar o sistema). Plataformas de banco de dados sem um fornecedor específico usar o ODBC mais genérico e provedores OLE DB, ambos incluídos com ADO.NET.

- Toda a comunicação com a fonte de dados externa ocorre através de um objeto de conexão ADO.NET, suporta o pool de conexão para aumentar a eficiência entre as consultas.
- Consultas SQL e instruções de gerenciamento de dados se envolvem em um objeto de comando antes de serem enviados para a fonte de dados. Os comandos podem incluir parâmetro opcional instâncias que

permitem que você chame procedimentos armazenados ou crie consultas fill-in-the-blan.

- O objeto padrão de armazenamento DataAdapter contém definições de consulta padrão para interagir com um banco de dados, eliminando o tédio de constantemente a necessidade de construir instruções SQL para cada registro que você deseja ler nem escrever, e ajudando a automatizar algumas tarefas relacionadas com o ADO.NET.
- O objeto DataReader fornece acesso somente de leitura rápida, com os resultados de uma consulta para aqueles momentos em que você só precisa obter seus dados rapidamente.

ADO.NET também inclui recursos que permitem salvar um DataSet inteiro como um arquivo XML e carrega-lo mais tarde.

4. ORM

Object Relationship Mapping é uma tentativa de mapear a noção de objeto relacional e do mundo para que eles possam falar uns com os outros de uma forma fácil. As maiorias das aplicações possuem um banco de dados por trás. Observando a maioria dos aplicativos percebemos que a aplicação se torna mais ou menos modelada em torno do modelo de dados. Na tecnologia de banco de dados, bancos de dados relacionais são os claros vencedores. Outras tecnologias de banco de dados veio e se foi.

4.1. BANCO DE DADOS RELACIONAIS

Banco de dados relacional foi criado em 1970, quando Edgar Frank Codd um pesquisador da IBM, escreveu um artigo descrevendo o processo. Desde então, bancos de dados relacionais têm crescido em popularidade para se tornar o padrão.

Banco de dados relacional é aquele que apresenta informações em tabelas com linhas e colunas. Uma tabela é referida como uma relação no sentido de que é uma coleção de objetos do mesmo tipo (linhas). Dados em uma tabela pode ser relacionado de acordo com chaves ou conceitos comuns, e a capacidade de recuperar dados de uma tabela relacionada é a base para o banco de dados relacional. Um Sistema de Gestão de Banco de Dados (SGBD) lida com a forma como os dados são armazenados, mantidos e recuperados. No caso de um banco de dados relacional, um Sistema de Gerenciamento de Banco de Dados Relacional (RDBMS) executa essas tarefas.

4.1.1. REGRAS DE INTEGRIDADE

Tabelas relacionais devem seguir certas regras de integridade para garantir que os dados estão disponíveis e sempre acessíveis. Todas as linhas de uma tabela relacional devem ser distintas. Se existirem linhas duplicadas, pode haver problemas na resolução do que duas seleções possíveis são a correta. Para a maioria dos SGBDs, o usuário pode especificar que as linhas

duplicadas não são permitidas, e se isso for feito, o DBMS irá impedir a adição de quaisquer linhas que duplicam uma linha existente.

A segunda regra de integridade do modelo relacional tradicional é que os valores da coluna não devem ser grupos ou repetir matrizes. Um terceiro aspecto da integridade dos dados envolve o conceito de um valor nulo. Um banco de dados cuida de situações em que os dados podem não estar disponíveis usando um valor nulo para indicar que um valor está em falta. Ele não equivale a um em branco ou zero. Um espaço em branco é considerado igual a outro em branco, um zero é igual a outro zero, mas dois valores nulos não são considerados iguais.

4.1.2. SQL

SQL é uma linguagem concebida para ser usada em bancos de dados relacionais. Há um conjunto de comandos SQL que é considerado padrão e é usado por todos os RDBMSs.

5. DESENVOLVIMENTO DOS SISTEMAS

Para desenvolvimento dos dois sistemas foi utilizado a ferramenta Visual Studio Community 2015 com Microsoft SQL Server 2014 SP1, todas as ferramentas utilizadas podem ser baixadas gratuitamente abaixo mostro um comparativo entre as versões do Visual Studio.

Figura 2 – Comparativo entre versões do Visual Studio.

	Visual Studio Community	Visual Studio Professional	Visual Studio Enterprise	Visual Studio Test Professional	Plataformas MSDN
Cenários de Uso Suportados	■■■□	■■■■	■■■■	■■■■	■■■■
Depuração e diagnóstico	■■■□	■■■□	■■■■	□□□□	□□□□
Ferramentas de teste	■□□□	■□□□	■■■■	■■□□	■■□□
Ambiente de desenvolvimento integrado	■■□□	■■■□	■■■■	□□□□	□□□□
Suporte à plataforma de desenvolvimento	■■■■	■■■■	■■■■	□□□□	□□□□
Arquitetura e modelagem	■□□□	■□□□	■■■■	□□□□	□□□□
Lab Management	□□□□	□□□□	■■■■	■■■■	■■■■
Recursos do Team Foundation Server	□□□□	■■■□	■■■■	■■■■	■■■■
Ferramentas de Colaboração	■■■■	■■■■	■■■■	■■■□	■■■□
Benefícios da colaboração de equipe	Incluído em todas as assinaturas				
Benefícios do Assinante	Incluído em assinaturas de nuvem e em assinaturas padrão anuais				
Benefícios do Visual Studio Dev Essentials	Gratuito para todos os desenvolvedores				

Fonte: Microsoft(2015).

Para o SQL Server foi instalado a ferramenta LocalDB na versão SQL Server 2014, para gerenciar o banco de dados foi instalado o SQL Server 2014 Management Studio, a ferramenta é totalmente em português e de fácil manuseio. Segue abaixo comparativo entre versões do SQL Server.

Figura 3 – Comparativo entre versões do SQL Server (2008 - 2014).

Recursos		SQL Server 2014	SQL Server 2012	SQL Server 2008 R2	SQL Server 2008
Desempenho	OLTP in-memory*	●			
	ColumnStore in-memory*	●	●		
	Extensão do pool de buffer para SSD	●			
	Administrador de recursos	●	●	●	●
Disponibilidade	AlwaysOn*	●	●		
	Suporte à virtualização aprimorado e migração dinâmica	●	●	●	
Segurança	Criptografia de dados transparente*	●	●	●	●
	Suporte à criptografia de backup	●			
	Auditoria refinada	●	●	●	●
	Separação de tarefas	●	●		
Preparação para nuvem	Backup para Microsoft Azure	●	●		
	Recuperação de desastres para Microsoft Azure	●			
	Images de VMs otimizadas na galeria do Microsoft Azure	●	●		
Gerenciamento e capacidade de programação	Distributed Replay	●	●		
	Gerenciamento baseado em políticas	●	●	●	●
	Capacidade de programação aprimorada	●	●	●	●

Fonte: Microsoft(2015).

Para gerenciar os dos pacotes instalados no sistema foi utilizado o Nuget gerenciador de pacotes para a plataforma de desenvolvimento da Microsoft.

5.1. CRIAR BANCOS DE DADOS PARA AS APLICAÇÕES

Para criação de scripts de banco de dados não será utilizado o gerenciador de criação de bancos de dados, tabelas, views e procedures da ferramenta Management Studio, todos os scripts vão ser criados manualmente para ser demostrada no decorrer do projeto.

Os nomes dos bancos de dados das aplicações ficaram o seguinte **DB_App_USE_ORM** e **DB_App_USE_ADONET** segue abaixo script de criação das Bases de dados.

Figura 4 – Criar Bancos de Dados.

```
-- Autor : Fernando Victor Pereira Santiago

-- Cria banco de dados DB_App_USE_ORM
Create database DB_App_USE_ORM;

-- Cria banco de dados DB_App_USE_ADONET
Create database DB_App_USE_ADONET;
```

Fonte: Interna (2015).

5.2. CRIAR TABELAS PARA AS APLICAÇÕES

Foi criado somente scripts de criação de tabelas para a aplicação com ADO Net, à aplicação com ORM controlara as criações de tabelas, chaves e etc para o sistema.

Segue abaixo a criação das tabelas **TAB_Pessoa** e **TAB_Pessoa_Fisica**.

Figura 5 – Criar Tabelas para o sistema com ADO Net.

```
-- Autor : Fernando Victor Pereira Santiago

Use DB_App_USE_ADONET;

-- Tabela Pessoa
CREATE TABLE TAB_Pessoa(
    IDPessoa INT IDENTITY(1,1) NOT NULL,
    Nome VARCHAR(80) NULL,
    Primary key (IDPessoa)
);

-- Tabela Pessoa Fisica
CREATE TABLE TAB_Pessoa_Fisica(
    IDPessoaFisica INT IDENTITY(1,1) NOT NULL,
    RG VARCHAR(8) NULL,
    CPF VARCHAR(11) NULL,
    Data_Nascimento DATETIME NULL,
    IDPessoa INT NOT NULL,
    Primary key (IDPessoaFisica)
);

-- Foreign Key da tabela Pessoa Fisica

ALTER TABLE TAB_Pessoa_Fisica ADD FOREIGN KEY(IDPessoa) REFERENCES TAB_Pessoa (IDPessoa);
```

Fonte: Interna(2015).

Para a criação de tabelas utilizando ORM foi criado no projeto **TCC.Dados** uma pasta chamada Mapeamentos, dentro dela foi criada uma classe para cada objeto que e o espelho do banco de dados. Segue abaixo a implementação das classes **PessoaMap** e **PessoaFisicaMap**.

Figura 6 – Classe PessoaMap.

```
public class PessoaMap : ClassMap<Pessoa>
{
    public PessoaMap()
    {
        // Nome da tabela
        Table("TAB_Pessoa");

        // Identificador da tabela
        Id(c => c.IDPessoa).GeneratedBy.Identity().Not.Nullable();

        // Campo nome da tabela
        Map(c => c.Nome).Column("Nome").Length(80).CustomType("AnsiString");
    }
}
```

Fonte: Interna (2015).

Figura 7 – Classe PessoaFisicaMap.

```
public class PessoaFisicaMap : ClassMap<PessoaFisica>
{
    public PessoaFisicaMap()
    {
        // Nome da tabela
        Table("TAB_Pessoa_Fisica");

        // Identificador da tabela
        Id(c => c.IDPessoaFisica).GeneratedBy.Identity().Not.Nullable();

        // Campos da tabela
        Map(c => c.RG).Length(8).CustomType("AnsiString");
        Map(c => c.CPF).Length(11).CustomType("AnsiString");
        Map(c => c.DataNascimento).Column("Data_Nascimento");

        // Chave estrangeira da tabela
        References(c => c.Pessoa).Not.Nullable().Column("IDPessoa").Cascade.All();
    }
}
```

Fonte: Interna (2015).

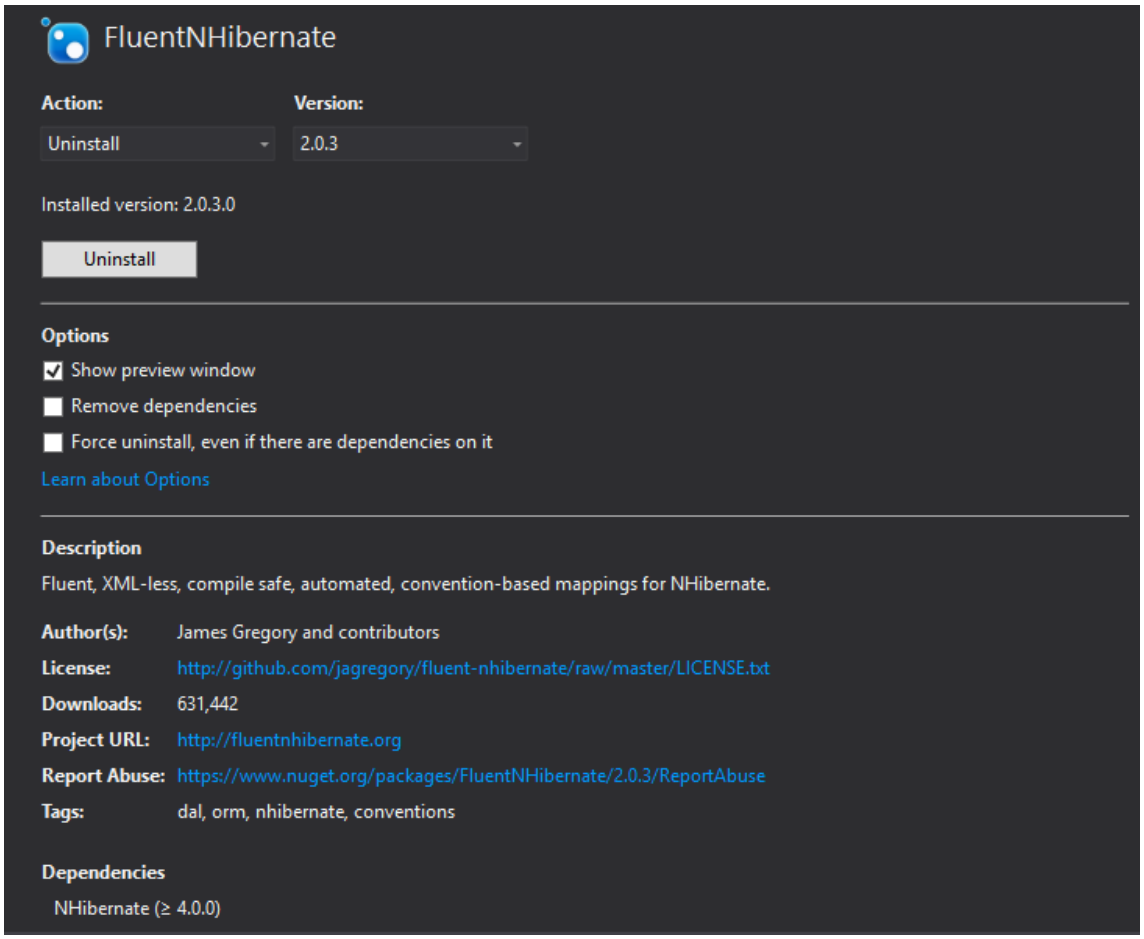
5.3. CRIAR PROJETOS

Para a criação dos projetos foi utilizado o gerenciador de novos projetos do Visual Studio, os projetos serão em C# com o Framework 4.5 da Microsoft, para depuração local será utilizado o servidor IIS que já vem previamente configurado.

Os projetos foram organizados dentro de Solutions Folder, foi organizado em três camadas Dados, Entidades e View. Onde dentro da camada **Entidades** possuem as classes que são o espelho do banco de dados, a camada **Dados** contem as classes responsáveis por persistir no banco de dados e a camada **View** contem o projeto com os arquivos CSS, Html, Imagens e etc.

A criação dos dois projetos seguiu o mesmo fluxo, só se difere que o projeto com o ORM foi instalado o pacote FluentNHibernate na versão estável 2.0.3 foi instalada através do Nuget. Na camada Dados do projeto ORM possui uma pasta chamada **Mapeamentos** onde contem as classes responsáveis por mapear os objetos para o banco de dados. As pastas Dao e Repositório seguem uma mesma ideia só se diferem no nome e no modo como são desenvolvidas as classes.

Figura 8 – Instalando pacote FluentNHibernate na versão 2.0.3.

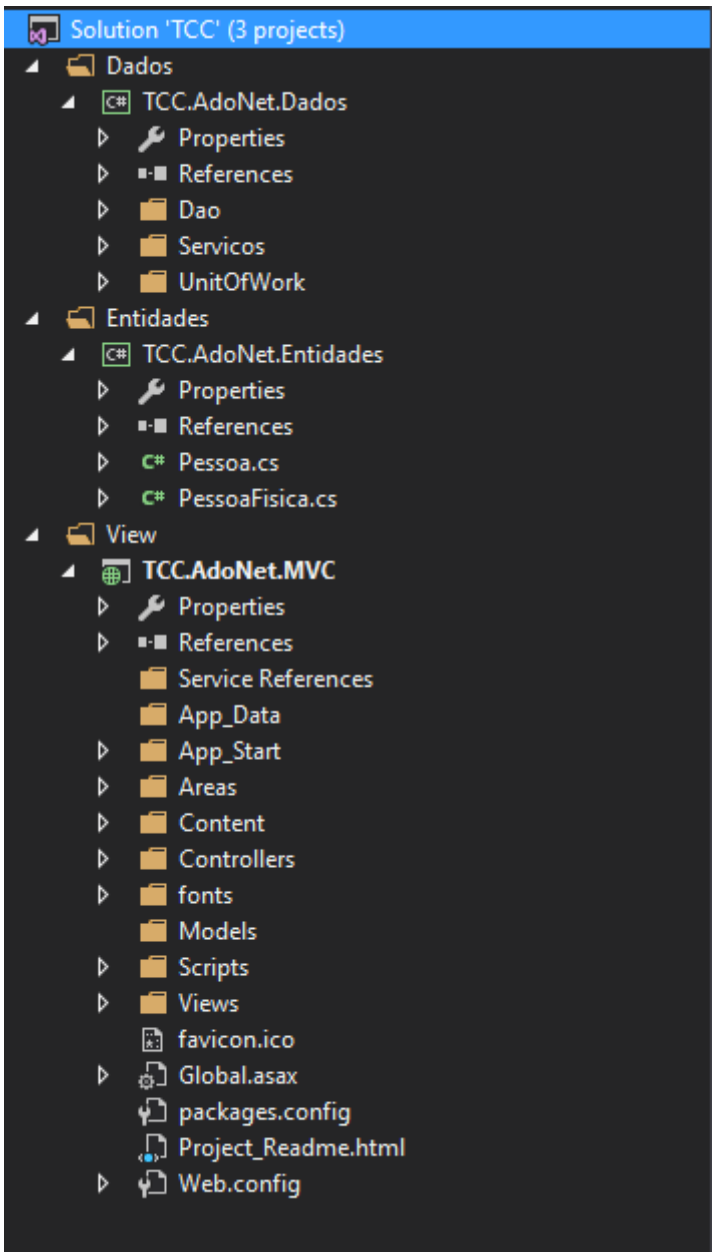


Fonte: Interna(2015).

5.3.1. CRIAR PROJETO ADO NET

Foi criado um projeto do tipo Blank Solution (projeto em branco) com o nome TCC. Foi criada os projetos **TCC.AdoNet.Dados e TCC.AdoNet.Entidades** do tipo Class Library respectivamente dentro das pastas Dados e Entidades. Foi criado também um projeto **TCC.AdoNet.MVC** do tipo Asp.Net Aplication dentro da pasta View. Segue abaixo estrutura do projeto.

Figura 9 – Estrutura do projeto ADO Net.

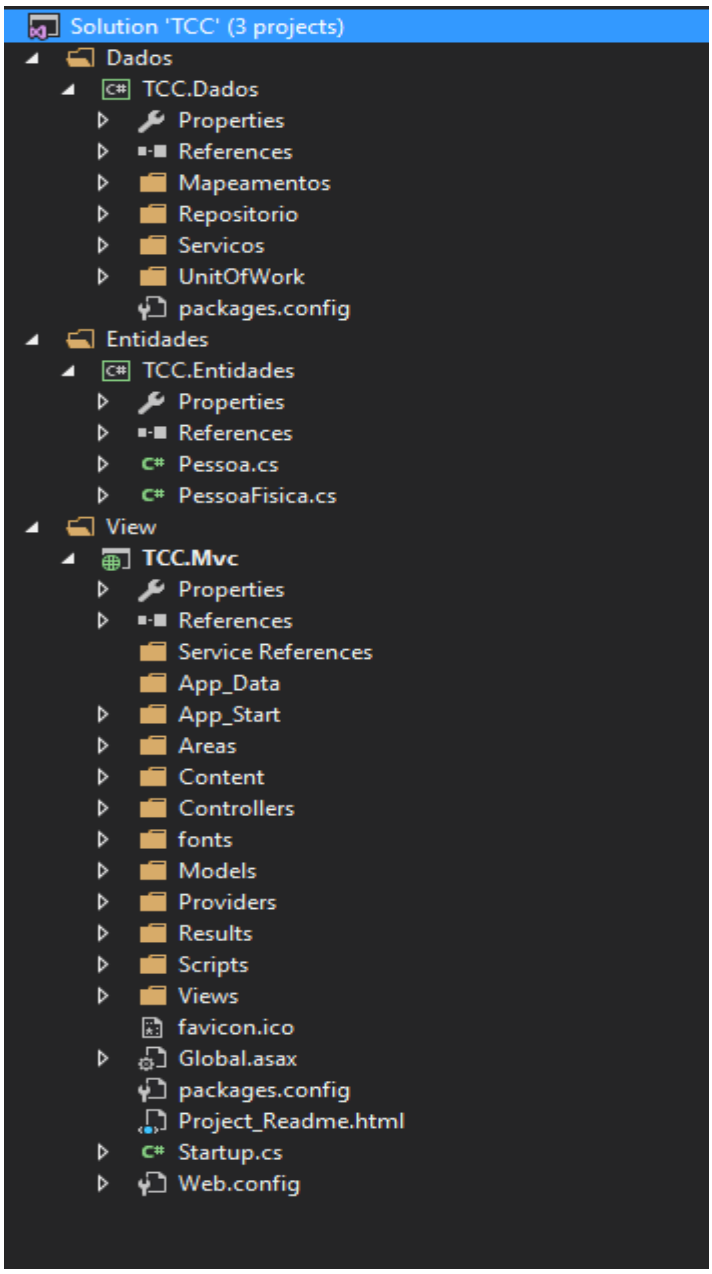


Fonte: Interna(2015).

5.3.2. CRIAR PROJETO ORM

Foi criado um projeto do tipo Blank Solution (projeto em branco) com o nome TCC. Foi criada os projetos **TCC.Dados e TCC.Entidades** do tipo Class Library respectivamente dentro das pastas Dados e Entidades. Foi criado também um projeto **TCC.MVC** do tipo Asp.Net Aplication dentro da pasta View. Segue abaixo estrutura do projeto.

Figura 10 – Estrutura do projeto ORM.



Fonte: Interna(2015).

5.4. CONEXÂO COM BANCO DE DADOS

A realização da conexão com o banco de dados foi feita com um padrão de projeto chamado **UnitOfWork** que mante uma lista de objetos afetados por uma transação e coordena a gravação de alterações e a resolução de problemas de concorrência. Foi aplicado o **UnitOfWork** nas duas aplicações com algumas particularidades em cada projeto.

5.4.1. CONEXÃO COM ADONET

A classe **UnitOfWork** responsável pela conexão com o banco de dados contem os seguintes propriedades privadas **_connection**, **_transaction** e **_disposed** que são usadas internamente na classe. A classe possui as propriedades publicas **Connection** e **Transaction**. Dentro do construtor da classe são iniciadas as propriedades **_connection**, **_transaction** e é aberta a conexão. Segue abaixo implementação do construtor da classe.

Figura 11 – Construtor da classe UnitOfWork.

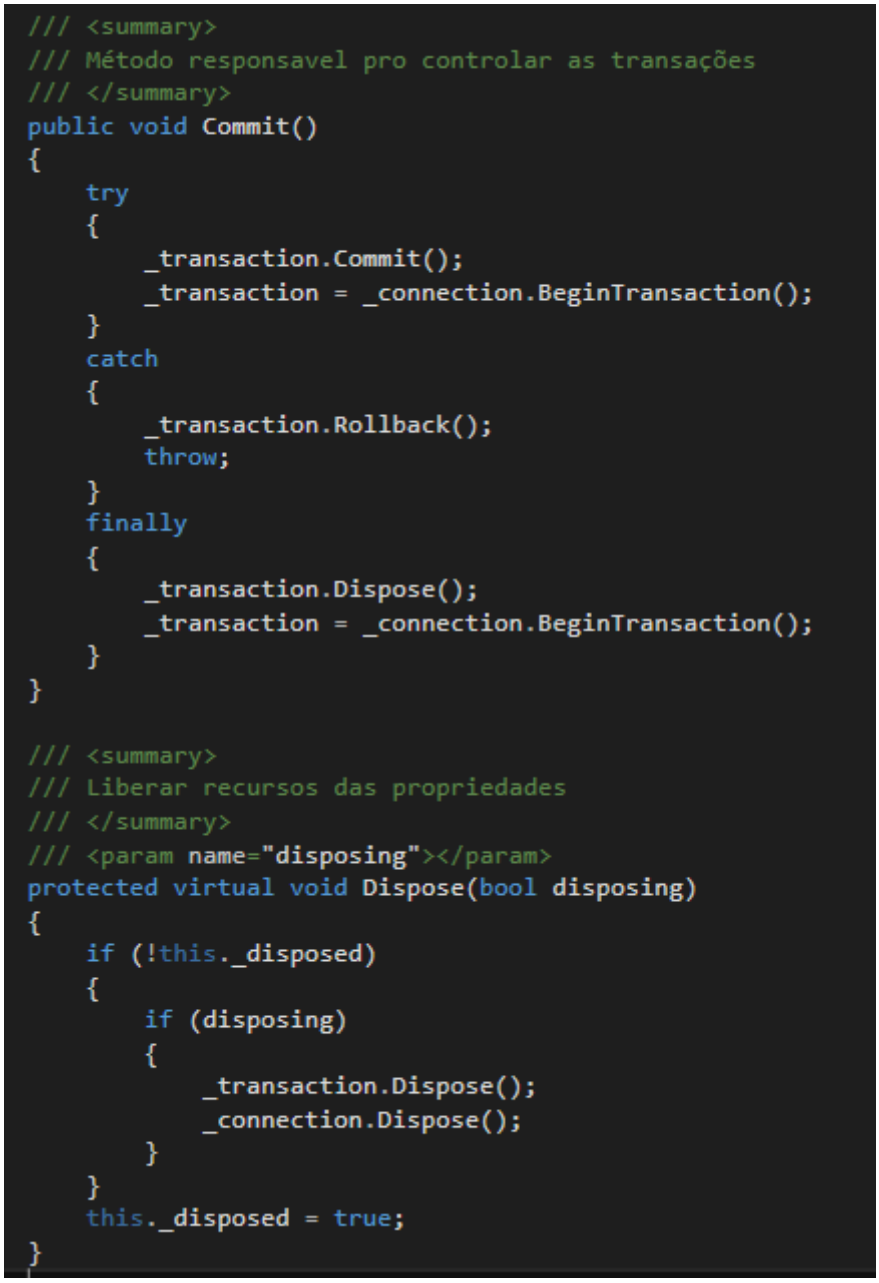
```
public class UnitOfWork : IUnitOfWork, IDisposable
{
    // Propriedades de Conexao, Transação e Disposed(Quando true liberar recursos)
    private readonly SqlConnection _connection;
    private SqlTransaction _transaction;
    private bool _disposed = false;

    // Encapsulo a propriedade Connection para retornar a propriedade _connection
    public SqlConnection Connection...

    // Encapsulo a propriedade Transaction para retornar a propriedade _transaction
    public SqlTransaction Transaction...

    /// <summary>
    /// Cria o contrutor da classe para iniciar as propriedades _connection, _transaction e abrir a conexao
    /// </summary>
    public UnitOfWork()
    {
        string connectionString = @"Persist Security Info=False;
                                    Integrated Security=true;
                                    Data Source=FERNANDO\SQLEXPRESS;
                                    Initial Catalog=DB_App_USE_ADONET;
                                    User ID=Fernando\FernandoPC;
                                    Password=;
                                    Language=Portuguese";
        _connection = new SqlConnection(connectionString);
        _connection.Open();
        _transaction = _connection.BeginTransaction();
    }
```

Fonte: Interna (2015).

A classe ainda possui dois métodos Commit e Dispose onde Commit controla as transações tanto para persistir alterações quanto para voltar

alterações feitas no banco, enquanto o método **Dispose** controla a liberação de recursos da classe. Segue abaixo implementação dos métodos.

Figura 12 – Métodos Commit e Dispose.



```
/// <summary>
/// Método responsavel pro controlar as transações
/// </summary>
public void Commit()
{
    try
    {
        _transaction.Commit();
        _transaction = _connection.BeginTransaction();
    }
    catch
    {
        _transaction.Rollback();
        throw;
    }
    finally
    {
        _transaction.Dispose();
        _transaction = _connection.BeginTransaction();
    }
}

/// <summary>
/// Liberar recursos das propriedades
/// </summary>
/// <param name="disposing"></param>
protected virtual void Dispose(bool disposing)
{
    if (!this._disposed)
    {
        if (disposing)
        {
            _transaction.Dispose();
            _connection.Dispose();
        }
    }
    this._disposed = true;
}
```

Fonte: Interna (2015).

5.4.2. CONEXÃO COM ORM

A classe UnitOfWork responsável pela conexão com o banco de dados contem os seguintes propriedades privadas _sessionFactory e _transaction. A classe também possui a propriedade publicas Session. Ela possui dois construtores um estático e um construtor publica. No estático e iniciado a configuração do banco e verificado se alguma classe mapeada foi alterada para refletir no banco. No construtor publico e aberto a conexão. Segue abaixo implementação dos construtores da classe.

Figura 13 – Construtores da classe UnitOfWork.

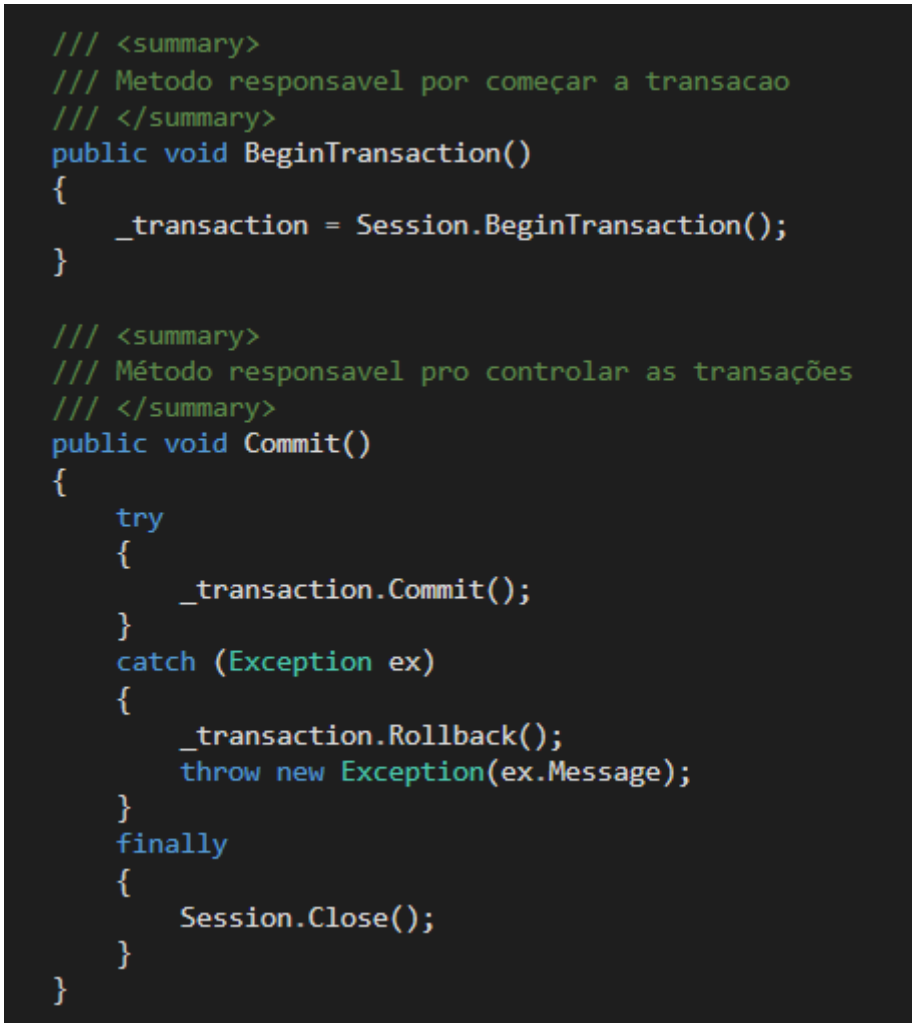
```
public class UnitOfWork : IUnitOfWork
{
    // Propriedades privadas ISessionFactory e ITransaction
    private static readonly ISessionFactory _sessionFactory;
    private ITransaction _transaction;

    // Propriedade publica ISession
    public ISession Session { get; private set; }

    /// <summary>
    /// Contrutor Statico para configurar sessao do NHibernate
    /// </summary>
    static UnitOfWork()
    {
        _sessionFactory = Fluently.Configure()
            .Database(
            MsSqlConfiguration.MsSql2012.ConnectionString(c => c
                .TrustedConnection()
                .Server(@"FERNANDO\SQLEXPRESS")
                .Database("App_USE_ORM")
                .Username(@"Fernando\FernandoPC")
            ).ShowSql())
            .Mappings(m => m.FluentMappings.AddFromAssembly(Assembly.GetExecutingAssembly()))
            .ExposeConfiguration(c => new SchemaUpdate(c).Execute(false, true))
            .BuildSessionFactory();
    }

    /// <summary>
    /// Contrutor publico para abrir a Sessao
    /// </summary>
    public UnitOfWork()
    {
        Session = _sessionFactory.OpenSession();
    }
```

Fonte: Interna (2015).

A classe possui dois métodos **BeginTransaction** e **Commit**, O método **BeginTransaction** e responsável por iniciar a transação enquanto o **Commit** e

responsável por controlar as alterações feitas no banco. Segue abaixo implementação dos métodos.

Figura 14 – Métodos BeginTransaction e Commit.



```
/// <summary>
/// Metodo responsavel por começar a transacao
/// </summary>
public void BeginTransaction()
{
    _transaction = Session.BeginTransaction();
}

/// <summary>
/// Método responsavel pro controlar as transações
/// </summary>
public void Commit()
{
    try
    {
        _transaction.Commit();
    }
    catch (Exception ex)
    {
        _transaction.Rollback();
        throw new Exception(ex.Message);
    }
    finally
    {
        Session.Close();
    }
}
```

Fonte: Interna (2015).

5.5. CRUD

Nessa parte que a utilização do ORM oferece vantagens significativas, pois no ORM podemos criar uma classe genérica que fara as operações do CRUD para todos os objetos que quisermos diminuindo muito o código.

5.5.1. METODOS CRUD DO SISTEMA ADONET

Para sistemas com ADONet para cada novo objeto que reflete o banco de dados terá que ser feitos os métodos manualmente escrevendo códigos SQL. Os diferentes bancos de dados relacionais possuem algumas diferenças na hora de se trabalhar com ele, ficando difícil a migração de um banco de dados para outro teria que ser testado o sistema inteiro se o mesmo não possuísse testes unitários. Um exemplo de diferença para outro e na forma como é retornado o código da ultima inserção feita no banco.

Quadro 1 – Diferença de como retornar um código do banco SQL Serve para Oracle.

SQL Server	Oracle
`insert into TAB_Pessoa (nome) OUTPUT INSERTED.IDPessoa values ('Fernando')`	`insert into TAB_Pessoa (nome) values('Fernando') returning IDPessoa into :outIDPessoa`

Fonte: Interna (2015).

Se fosse ter que migrar do SQL Server para Oracle teria que rever todos os códigos SQL.

Foi criado uma classe **PessoaDao** responsável por manipular os dados da pessoa. Foram criados os métodos **Add**, **Get**, **GetAll**, **Remove** e **Update**. Segue abaixo classe com seus métodos.

Figura 15 – Classe PessoaDao com seus Métodos.

```
/// <summary> Classe responsavel por manipular dados no banco relacionado a pessoa
public class PessoaDao
{
    /// <summary> Metodo que adiciona uma pessoa fisica ao banco de dados
    public void Add(PessoaFisica pessoaFisica)...

    /// <summary> Metodo que retorna uma pessoa fisica do banco pelo id
    public PessoaFisica Get(int id)...

    /// <summary> Método retorna todas as Pessoas Fisicas cadastradas no Banco de Dados
    public List<PessoaFisica> GetAll()...

    /// <summary> Método que remove uma pessoa Fisica do Banco de dados
    public void Remove(int id)...

    /// <summary> Método que remove uma Pessoa Fisica do Banco de Dados
    public void Update(PessoaFisica pessoaFisica)...

    public void Dispose()...
}
```

Fonte: Interna (2015).

Método que adiciona uma pessoa física ao banco de dados.

Figura 16 – Método Add da classe PessoaDao.

```
/// <summary> Metodo que adiciona uma pessoa fisica ao banco de dados
public void Add(PessoaFisica pessoaFisica)
{
    string sql = string.Empty;
    SqlCommand cmd = null;

    try
    {
        using (var conexao = new UnitOfWork.UnitOfWork())
        {
            sql = "insert into TAB_Pessoa (nome) OUTPUT INSERTED.IDPessoa values (@nome)";

            cmd = new SqlCommand(sql, conexao.Connection);
            cmd.Transaction = conexao.Transaction;
            cmd.Parameters.AddWithValue("@nome", pessoaFisica.Pessoa.Nome);
            pessoaFisica.Pessoa.IDPessoa = (Int32)cmd.ExecuteScalar();

            sql = string.Empty;
            sql = "insert into TAB_Pessoa_Fisica (RG, CPF, Data_Nascimento, IDPessoa) values(@rg, @cpf, @dataNascimento, @idPessoa)";

            cmd = new SqlCommand(sql, conexao.Connection);
            cmd.Transaction = conexao.Transaction;
            cmd.Parameters.AddWithValue("@rg", pessoaFisica.RG);
            cmd.Parameters.AddWithValue("@cpf", pessoaFisica.CPF);
            cmd.Parameters.AddWithValue("@dataNascimento", pessoaFisica.DataNascimento);
            cmd.Parameters.AddWithValue("@idPessoa", pessoaFisica.Pessoa.IDPessoa);
            cmd.ExecuteNonQuery();

            conexao.Commit();
        }
    }
    catch (Exception)
    {
        throw;
    }
}
```

Fonte: Interna (2015).

Método que retorna uma pessoa física cadastrada no banco de dados pelo id.

Figura 17 – Método Get da classe PessoaDao.

```
/// <summary> Metodo que retorna uma pessoa fisica do banco pelo id
public PessoaFisica Get(int id)
{
    // variaveis pessoaFisica sql e cmd
    PessoaFisica pessoaFisica = new PessoaFisica();
    string sql = string.Empty;
    SqlCommand cmd = null;
    SqlDataReader dr = null;

    try
    {
        using (var conexao = new UnitOfWork.UnitOfWork())
        {
            sql = @"select IDPessoaFisica, RG, CPF, Data_Nascimento, IDPessoa
                from TAB_Pessoa_Fisica where IDPessoaFisica = @IDPessoaFisica";
            cmd = new SqlCommand(sql, conexao.Connection);
            cmd.Parameters.AddWithValue("@IDPessoaFisica", id);
            cmd.Transaction = conexao.Transaction;

            dr = cmd.ExecuteReader();
            while (dr.Read())
            {
                pessoaFisica.IDPessoaFisica = (Int32)dr["IDPessoaFisica"];
                pessoaFisica.RG = (string)dr["RG"];
                pessoaFisica.DataNascimento = (DateTime)dr["Data_Nascimento"];
                pessoaFisica.Pessoa.IDPessoa = (Int32)dr["IDPessoa"];
                pessoaFisica.CPF = (string)dr["CPF"];
            }

            sql = string.Empty;
            sql = "select nome from TAB_Pessoa where IDPessoa = @IDPessoa";
            cmd = new SqlCommand(sql, conexao.Connection);
            cmd.Parameters.AddWithValue("@IDPessoa", pessoaFisica.Pessoa.IDPessoa);
            cmd.Transaction = conexao.Transaction;

            dr.Close();
            dr = cmd.ExecuteReader();
            while (dr.Read())
            {
                pessoaFisica.Pessoa.Nome = (string)dr["nome"];
            }

            dr.Close();

            return pessoaFisica;
        }
    }
    catch (Exception)
    {
        throw;
    }
}
```

Fonte: Interna (2015).

Método que retorna todas as pessoas físicas cadastradas no banco de dados.

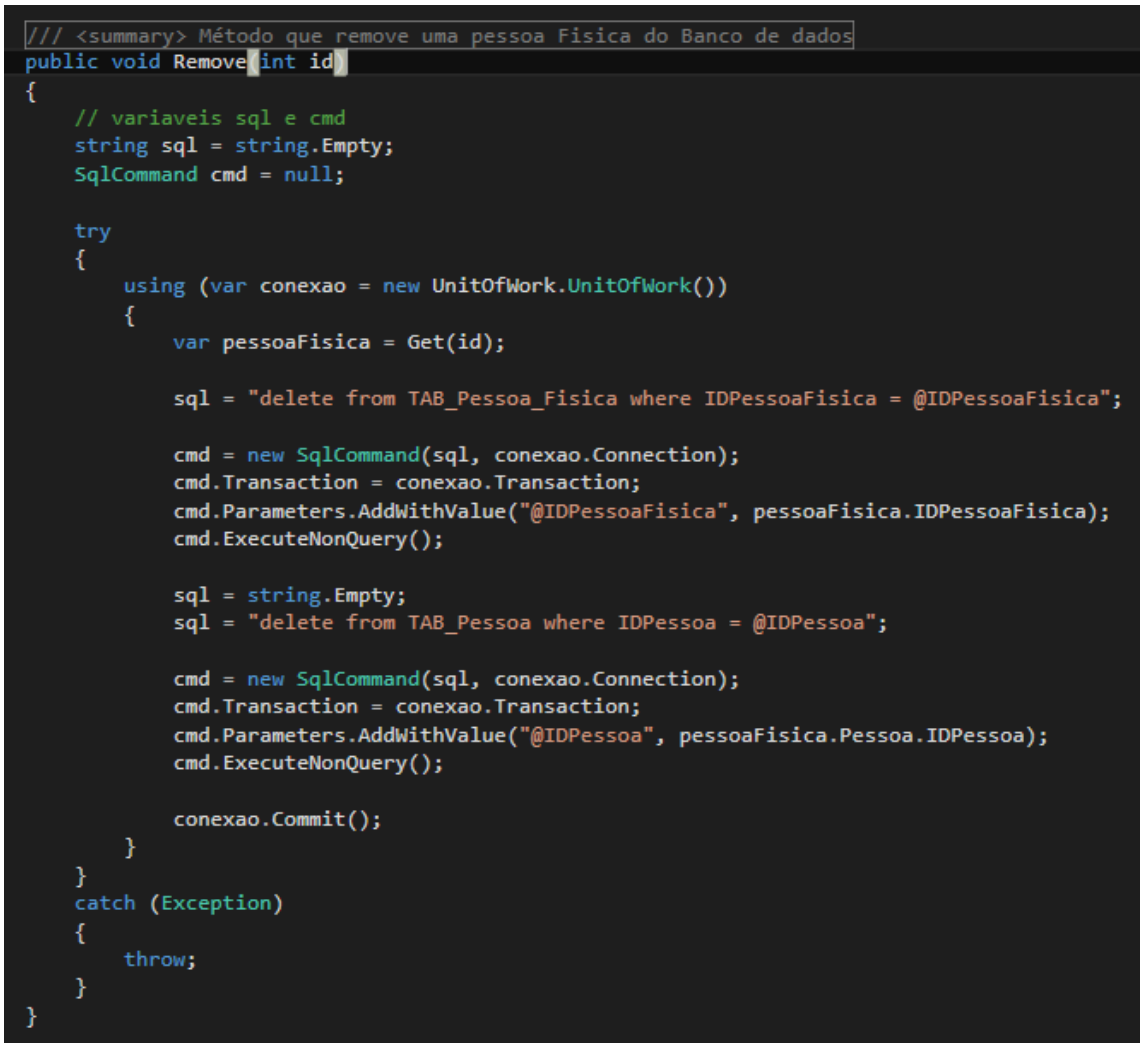
Figura 18 – Método GetAll da classe PessoaDao.

```
/// <summary> Método retorna todas as Pessoas Fisicas cadastradas no Banco de Dados
public List<PessoaFisica> GetAll()
{
    List<PessoaFisica> lstPessoaFisica = new List<PessoaFisica>();
    string sql = string.Empty;
    SqlCommand cmd = null;
    SqlDataReader dr = null;
    try
    {
        using (var conexao = new UnitOfWork.UnitOfWork())
        {
            sql = @"select IDPessoaFisica, RG, CPF, Data_Nascimento, IDPessoa
                from TAB_Pessoa_Fisica";
            cmd = new SqlCommand(sql, conexao.Connection);
            cmd.Transaction = conexao.Transaction;

            dr = cmd.ExecuteReader();
            while (dr.Read())
            {
                var pessoaFisica = new PessoaFisica();
                pessoaFisica.IDPessoaFisica = (Int32)dr["IDPessoaFisica"];
                pessoaFisica.RG = (string)dr["RG"];
                pessoaFisica.DataNascimento = (DateTime)dr["Data_Nascimento"];
                pessoaFisica.Pessoa.IDPessoa = (Int32)dr["IDPessoa"];
                pessoaFisica.CPF = (string)dr["CPF"];
                lstPessoaFisica.Add(pessoaFisica);
            }
            foreach (var item in lstPessoaFisica)
            {
                sql = string.Empty;
                sql = "select nome from TAB_Pessoa where IDPessoa = @IDPessoa";

                cmd = new SqlCommand(sql, conexao.Connection);
                cmd.Parameters.AddWithValue("@IDPessoa", item.Pessoa.IDPessoa);
                cmd.Transaction = conexao.Transaction;

                dr.Close();
                dr = cmd.ExecuteReader();
                while (dr.Read())
                {
                    item.Pessoa.Nome = (string)dr["nome"];
                }
            }
            dr.Close();
            return lstPessoaFisica;
        }
    }
    catch (Exception)
    {
        throw;
    }
}
```

Fonte: Interna (2015).

Método que remove uma pessoa física cadastrada no banco de dados.

Figura 19 – Método Remove da classe PessoaDao.

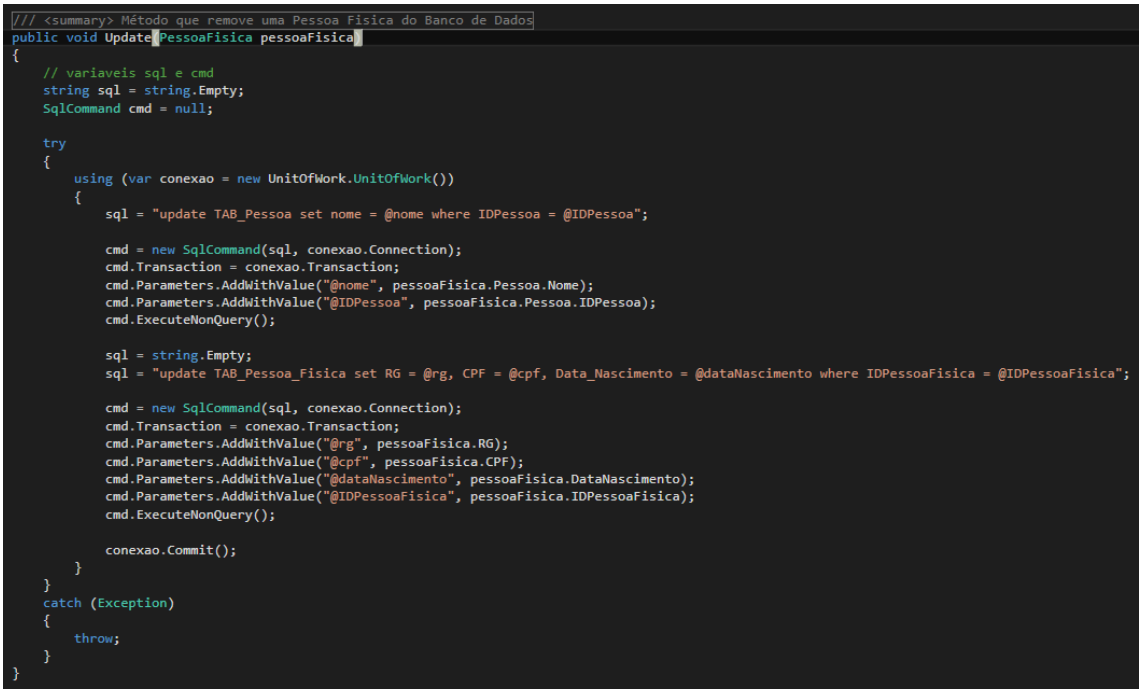


```
/// <summary> Método que remove uma pessoa Fisica do Banco de dados
public void Remove(int id)
{
    // variaveis sql e cmd
    string sql = string.Empty;
    SqlCommand cmd = null;

    try
    {
        using (var conexao = new UnitOfWork.UnitOfWork())
        {
            var pessoaFisica = Get(id);

            sql = "delete from TAB_Pessoa_Fisica where IDPessoaFisica = @IDPessoaFisica";

            cmd = new SqlCommand(sql, conexao.Connection);
            cmd.Transaction = conexao.Transaction;
            cmd.Parameters.AddWithValue("@IDPessoaFisica", pessoaFisica.IDPessoaFisica);
            cmd.ExecuteNonQuery();

            sql = string.Empty;
            sql = "delete from TAB_Pessoa where IDPessoa = @IDPessoa";

            cmd = new SqlCommand(sql, conexao.Connection);
            cmd.Transaction = conexao.Transaction;
            cmd.Parameters.AddWithValue("@IDPessoa", pessoaFisica.Pessoa.IDPessoa);
            cmd.ExecuteNonQuery();

            conexao.Commit();
        }
    }
    catch (Exception)
    {
        throw;
    }
}
```

Fonte: Interna (2015).

Método que atualiza uma pessoa física cadastrada no banco de dados.

Figura 20 – Método Update da classe PessoaDao.



```
/// <summary> Método que remove uma Pessoa Fisica do Banco de Dados
public void Update(PessoaFisica pessoaFisica)
{
    // variaveis sql e cmd
    string sql = string.Empty;
    SqlCommand cmd = null;

    try
    {
        using (var conexao = new UnitOfWork.UnitOfWork())
        {
            sql = "update TAB_Pessoa set nome = @nome where IDPessoa = @IDPessoa";

            cmd = new SqlCommand(sql, conexao.Connection);
            cmd.Transaction = conexao.Transaction;
            cmd.Parameters.AddWithValue("@nome", pessoaFisica.Pessoa.Nome);
            cmd.Parameters.AddWithValue("@IDPessoa", pessoaFisica.Pessoa.IDPessoa);
            cmd.ExecuteNonQuery();

            sql = string.Empty;
            sql = "update TAB_Pessoa_Fisica set RG = @rg, CPF = @cpf, Data_Nascimento = @dataNascimento where IDPessoaFisica = @IDPessoaFisica";

            cmd = new SqlCommand(sql, conexao.Connection);
            cmd.Transaction = conexao.Transaction;
            cmd.Parameters.AddWithValue("@rg", pessoaFisica.RG);
            cmd.Parameters.AddWithValue("@cpf", pessoaFisica.CPF);
            cmd.Parameters.AddWithValue("@dataNascimento", pessoaFisica.DataNascimento);
            cmd.Parameters.AddWithValue("@IDPessoaFisica", pessoaFisica.IDPessoaFisica);
            cmd.ExecuteNonQuery();

            conexao.Commit();
        }
    }
    catch (Exception)
    {
        throw;
    }
}
```

Fonte: Interna (2015).

Para cada classe Dao e criada dentro do projeto **TCC.AdoNet.Dados** na pasta **Servicos** uma classe que é a ponte entre as classes Dao e a View. A classe possui os mesmos métodos da Dao, os métodos da classe da pasta serviços somente chamam os métodos da classe Dao. Segue abaixo a implementação da classe **PessoaBus.**

Figura 21 – Classe PessoaBus.

```
public class PessoaBus
{
    private PessoaDao _pessoaDao;

    public PessoaBus()...

    public void Add(PessoaFisica pessoaFisica)
    {
        _pessoaDao.Add(pessoaFisica);
    }

    public PessoaFisica Get(int id)
    {
        return _pessoaDao.Get(id);
    }

    public List<PessoaFisica> GetAll()
    {
        return _pessoaDao.GetAll();
    }

    public void Remove(int id)
    {
        _pessoaDao.Remove(id);
    }

    public void Update(PessoaFisica pessoaFisica)
    {
        _pessoaDao.Update(pessoaFisica);
    }

}
```

Fonte: Interna (2015).

5.5.2. METODOS CRUD DO SISTEMA ORM

Para os CRUD no sistema com ORM foi criada a classe **Repositorio** que é uma classe genérica, ou seja, qualquer classe que seja a espelho do banco de dados já terão todos os métodos do CRUD pronto. Se for preciso de outro método além do CRUD pode ser feito com **LINQ to SQL**. Foi criada a classe **PessoaService** que somente chama os métodos da classe **Repositorio**. Segue implementação da classe **Repositorio.**

Figura 22 – Classe Repositorio.

```
/// <summary>
/// Repositorio Generico para simples Crud no sistema
/// </summary>
/// <typeparam name="T"></typeparam>
public class Repositorio<T> : IRepositorio<T> where T : class
{
    private TCC.Dados.UnitOfWork.UnitOfWork _unitOfWork;

    /// <summary>
    /// Contrutor com Injecao de Dependencia do IUnitOfWork para poder ser usada as propriedades publicas do UnitOfWork
    /// </summary>
    /// <param name="unitOfwork"></param>
    public Repositorio(TCC.Dados.UnitOfWork.IUnitOfWork unitOfwork)
    {
        _unitOfWork = (TCC.Dados.UnitOfWork.UnitOfWork)unitOfwork;
    }

    protected ISession Session { get { return _unitOfWork.Session; } }
```

Fonte: Interna (2015).

O modo como e feito a manipulação de dados com **NHibernate** e muito simples, para uma aplicação SPA (Single Page Application) o desenvolvimento com **NHibernate** fica muito produtivo. Segue abaixo os métodos genéricos do CRUD.

Figura 23 – Métodos CRUD da classe Repositorio.

```
/// <summary> Metodo responsavel por salvar um Objeto Generico
public void Add(T obj)
{
    Session.Save(obj);
}

/// <summary> Metodo responsavel por atualizar um Objeto Generico
public void Update(T obj)
{
    Session.Update(obj);
}

/// <summary> Metodo responsavel por remover um Objeto Generico por ID
public void Remove(int id)
{
    Session.Delete(Get(id));
}

/// <summary> Metodo responsavel por retornar um Objeto Generico
public T Get(int id)
{
    return Session.Get<T>(id);
}

/// <summary> Metodo que retorna uma Lista de Objetos Genericos
public IQueryable<T> GetAll()
{
    return Session.Query<T>();
}
```

Fonte: Interna (2015).

A classe **PessoaService** simplesmente chama os métodos da classe **Repositorio** única coisa que precisa ser destacada e no construtor da classe que é instanciado a classe **Repositorio** do tipo **PessoaFisica** para o **NHibernate** entender o objeto que esta fazendo as alterações. Segue implementação da classe **PessoaService.**

Figura 24 – Classe PessoaService.

```
public class PessoaService : IPessoaService
{
    private IRepositorio<PessoaFisica> _pessoaRepositorio;

    public PessoaService(IRepositorio<PessoaFisica> pessoaRepositorio)
    {
        _pessoaRepositorio = pessoaRepositorio;
    }

    public void Add(PessoaFisica pessoa)
    {
        _pessoaRepositorio.Add(pessoa);
    }

    public PessoaFisica Get(int id)
    {
        return _pessoaRepositorio.Get(id);
    }

    public IList<PessoaFisica> GetAll()
    {
        return _pessoaRepositorio.GetAll().ToList();
    }

    public void Remove(int id)
    {
        _pessoaRepositorio.Remove(id);
    }

    public void Update(PessoaFisica pessoa)
    {
        _pessoaRepositorio.Update(pessoa);
    }
}
```

Fonte: Interna (2015).

6. DESEMPENHO DO CRUD NOS SISTEMAS

Para medir o desempenho dos sistemas em relação às operações CRUD será utilizado a classe **Stopwach** (Cronometro) da namespace **System.Diagnostics.** Segue exemplo da implementação da funcionalidade para medir o desempenho do método Add utilizando a classe Stopwach.

Figura 25 – Implementação de método para medir desempenho CRUD.

```
PessoaBus pesBus = new PessoaBus();

// Estancia a classe stopWatch
Stopwatch stopWatch = new Stopwatch();
// Inicia o cronometro
stopWatch.Start();

// Percorre 500 vezes salvando uma Pessoa Fisica
for (int i = 0; i < 500; i++)
{
    PessoaFisica pessoaFisica = new PessoaFisica();
    pessoaFisica.CPF = "01345678912";
    pessoaFisica.DataNascimento = new DateTime(1990, 12, 17);
    pessoaFisica.RG = "12345678";
    pessoaFisica.Pessoa.Nome = "Fernando Victor Pereira Santiago";
    pesBus.Add(pessoaFisica);
}

// Quando termina para o cronometro
stopWatch.Stop();
// Cria um variavel TimeSpan para pegar o tempo gasto na operacao
TimeSpan ts = stopWatch.Elapsed;

// Formato o tempo para melhor vizualizacao
string tempoGasto = String.Format("{0:00}:{1:00}:{2:00}.{3:00}",
    ts.Hours, ts.Minutes, ts.Seconds,
    ts.Milliseconds / 10);

// Coloca numa viewBag ver na tela do navegador
ViewBag.Tempo = "Tempo gasto " + tempoGasto;
```

Fonte: Interna (2015).

O desempenho será mostrado em um quadro onde terá a quantidade de vezes que foi chamada a operação e o tempo gasto para realizar a quantidade

informada de operações. O tempo vira no seguinte formato Hora : Minuto : Segundo : Miles Segundo.

Desempenho do Sistema com ADONET

Quadro 2 – Desempenho dos métodos CRUD no sistema com ADONET.

Qtd	Add	Get	Update	Remove	Qtd	GetAll
500	00:00:00.87	00:00:00.17	00:00:01.11	00:00:01.79	**5**	00:00:03.09
1000	00:00:01.08	00:00:00.33	00:00:01.31	00:00:02.13	**10**	00:00:06.16
1500	00:00:01.60	00:00:00.46	00:00:01.62	00:00:02.48	**15**	00:00:09.24
2000	00:00:02.56	00:00:00.62	00:00:02.29	00:00:02.68	**50**	00:00:30.23
2500	00:00:02.60	00:00:00.77	00:00:03.17	00:00:03.38	**100**	00:01:01.66

Fonte: Interna (2015).

Desempenho do Sistema com ORM

Quadro 3 – Desempenho dos métodos CRUD no sistema com ORM.

Qtd	Add	Get	Update	Remove	Qtd	GetAll
500	00:00:00.22	00:00:00.14	00:00:00.33	00:00:00.34	**5**	00:00:01.04
1000	00:00:00.40	00:00:00.29	00:00:00.51	00:00:00.44	**10**	00:00:01.17
1500	00:00:00.79	00:00:00.37	00:00:00.68	00:00:00.47	**15**	00:00:01.52
2000	00:00:00.75	00:00:00.50	00:00:00.86	00:00:00.59	**50**	00:00:04.25
2500	00:00:01.08	00:00:00.60	00:00:01.15	00:00:00.67	**100**	00:00:07.91

Fonte: Interna (2015).

O testes de desempenho do método **GetAll** foi executado após o método **Update** para não ser removido todas as informações do banco. Foi feita em menos quantidades o método **GetAll** pelo fato de ter 7500 em cada tabela, ficando um tempo longo de espera entre um teste e outro.

Como pode ser visto nos quadros 2 e 3 o sistema com ORM e mais rápido ate três vezes em quase todos os métodos, o único método que os tempos estão próximo um do outro é o método **Get** ainda assim o ORM e mais rápido mais não com tanta diferença.

7. MIGRAR SISTEMAS PARA BANCO DE DADOS POSTGRESQL

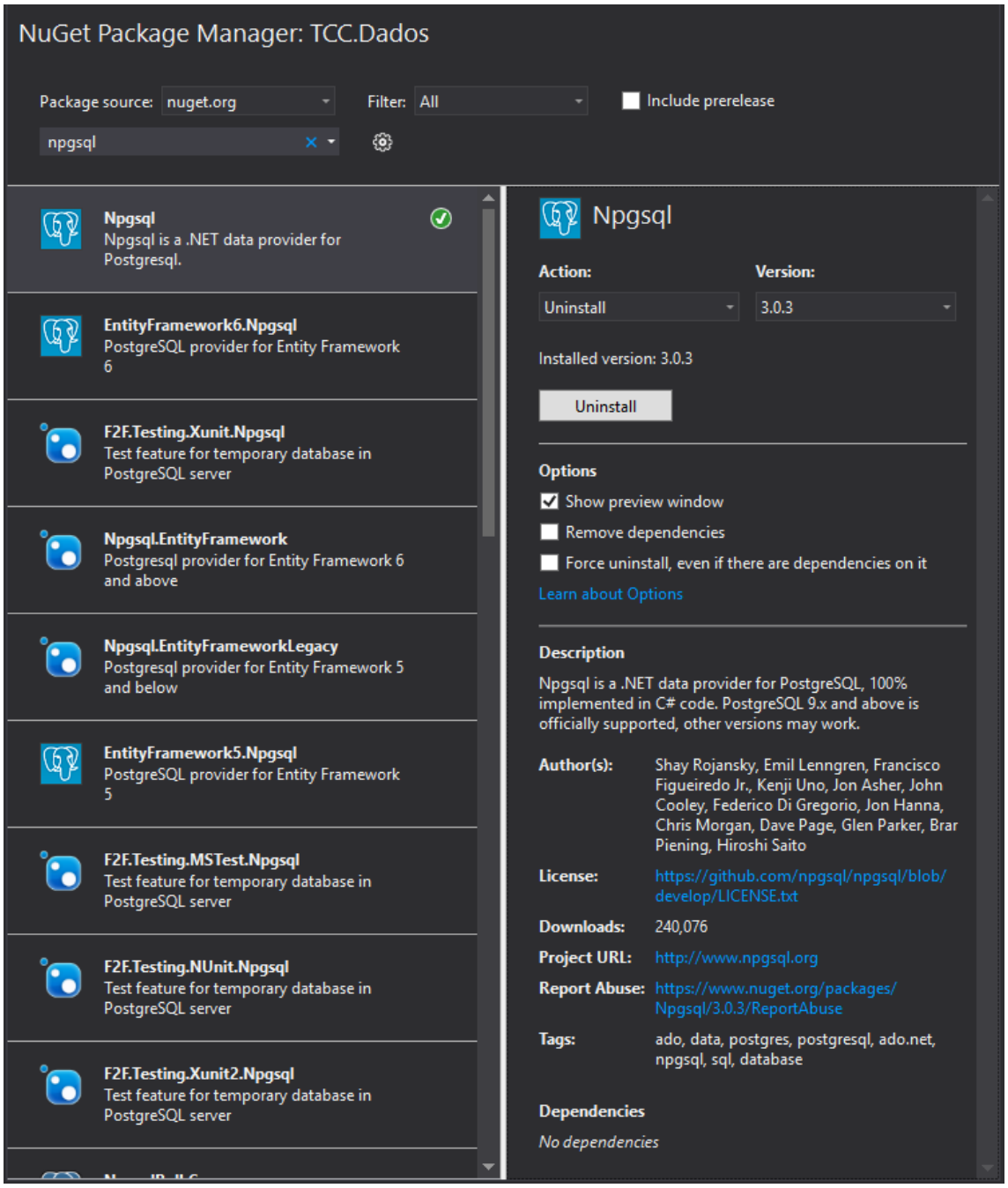
O processo da migração pode ser bastante complexo, apenas alterarei o banco de dados utilizado na aplicação não migrarei informações. Para além da dificuldade de transferir informação entre os dois sistemas gestores de bases de dados, também influirão muito na complexidade do problema o tipo de dados das tabelas que estamos a utilizar. Por exemplo, as datas, os campos numéricos com decimais ou os booleanos podem dar problemas ao passar de um sistema a outro porque podem armazenar-se de maneiras diferentes ou nos caso dos números, com uma precisão diferente.

A banco de dados escolhido para a migração foi o **PostGreSQL** na versão 9.4. Foi escolhido esse banco por já o ter instalado e configurado na minha maquina. Este banco também e o ideal para o teste de migração pelo mesmo não possuir **IDENTITY** do SQL Server, o **PostGreSQL** utiliza **Sequence** para fazer Auto Incremento.

Migrar Sistema com ORM

Para realizar a migração do sistema com ORM foi bastante simples, somente tive que instalar o pacote **Npgsql** na versão estável 3.0.3 e alterar a classe **UnitOfWork** mudando a configuração do banco de dados. Todos os métodos do CRUD funcionaram normalmente. Segue abaixo alterações que precisaram ser feitas para funcionar com PostgreSQL.

Figura 26 – Instalando o pacote npgsql no sistema com ORM.



Fonte: Interna (2015).

Na classe **UnitOfWork** apenas precisei alterar a configuração que monta a string de conexão de **MsSqlConfiguration** para **PostgreSQLConfiguration** e colocar o Host, Port, Username, Password e Database. Segue abaixo como ficou o construtor estático da classe **UnitOfWork**.

Figura 27 – Classe UnitOfWork configurada para PostGreSQL.

```
/// <summary>
/// Contrutor Statico para configurar sessao do NHibernate com PostgreSQL
/// </summary>
static UnitOfWork()
{
    _sessionFactory = Fluently.Configure()
        .Database(
        PostgreSQLConfiguration.PostgreSQL82.ConnectionString(c => c
            .Host("localhost")
            .Port(5432)
            .Username("postgres")
            .Password("123123")
            .Database("DB_App_USE_ORM")
        ).ShowSql())
        .Mappings(m => m.FluentMappings.AddFromAssembly(Assembly.GetExecutingAssembly()))
        .ExposeConfiguration(c => new SchemaUpdate(c).Execute(false, true))
        .BuildSessionFactory();
}

///// <summary>
///// Contrutor Statico para configurar sessao do NHibernate com SQL Server
///// </summary>
//static UnitOfWork()
//{
//    _sessionFactory = Fluently.Configure()
//        .Database(
//        MsSqlConfiguration.MsSql2012.ConnectionString(c => c
//            .TrustedConnection()
//            .Server(@"FERNANDO\SQLEXPRESS")
//            .Database("DB_App_USE_ORM")
//            .Username(@"Fernando\FernandoPC")
//        ).ShowSql())
//        .Mappings(m => m.FluentMappings.AddFromAssembly(Assembly.GetExecutingAssembly()))
//        .ExposeConfiguration(c => new SchemaUpdate(c).Execute(false, true))
//        .BuildSessionFactory();
//}
```

Fonte: Interna (2015).

Migrar Sistema com ADONet

Para realizar a migração do sistema com ADO Net teve que ser alterado o scritp de criação de tabelas para ser feito o AutoIncrement dos Ids das tabelas. Para esta migração tive que criar a classe **UnitOfWorkPG** pois teria que ser alterada varias coisas na classe. Para esta migração que só possui uma classe de acesso a dados foi meio desgastante, pois teve que ser mudado todos os métodos onde possuía **SqlCommand** teve que mudar para **NpgsqlCommand**, onde possuía **SqlDataReader** teve que mudar para **NpgsqlDataReader**. O único método que precisou alterar o script SQL, foi o método **Add** que precisa retornar o ID inserido.

A classe **UnitOfWorkPG** foi copiada tudo que tinha na classe **UnitOfWork,** foi mudando as propriedades que antes eram do tipo **SqlConnection** e **SqlTransaction** para respectivamente **NpgsqlConnection** e **NpgsqlTransaction.** Foi também mudado o construtor da classe para alterar a string de conexão. Segue como ficou a implementação da nova classe.

Figura 28 – Classe UnitOfWorkPG configurada para PostGreSQL.

```
public class UnitOfWorkPG : IDisposable
{
    // Propriedades de Conexao, Transação e Disposed(Quando true liberar recursos)
    private readonly NpgsqlConnection _connection;
    private NpgsqlTransaction _transaction;
    private bool _disposed = false;

    // Encapsulo a propriedade Connection para retornar a propriedade _connection
    public NpgsqlConnection Connection
    {
        get { return _connection; }
    }

    // Encapsulo a propriedade Transaction para retornar a propriedade _transaction
    public NpgsqlTransaction Transaction
    {
        get { return _transaction; }
    }

    /// <summary> Cria o contrutor da classe para iniciar as propriedades _connection, _transaction e abrir a conexao
    public UnitOfWorkPG()
    {
        string connectionString = @"Server=localhost;
                                    Port=5432;
                                    User Id=postgres;
                                    Password=123123;
                                    Database=DB_App_USE_ADONET";
        _connection = new NpgsqlConnection(connectionString);
        _connection.Open();
        _transaction = _connection.BeginTransaction();
    }
```

Fonte: Interna (2015).

Os scripts de criação da tabela também sofrerão alteração, pois para realizar o auto incremento toda tabela deve possuir uma **Sequence**. Segue como ficou o script de criação das tabelas.

Figura 29 – Script de criação de tabelas para o PostgreSQL.

```
-- Autor : Fernando Victor Pereira Santiago

Use DB_App_USE_ADONET;

-- Tabela Pessoa
CREATE SEQUENCE seq_TAB_Pessoa INCREMENT 1 MINVALUE 1 MAXVALUE 9223372036854775807 START 1 CACHE 1;

CREATE TABLE TAB_Pessoa(
        IDPessoa INT NOT NULL DEFAULT NEXTVAL('seq_TAB_Pessoa'),
        Nome VARCHAR(80) NULL,
        Primary key (IDPessoa)
);

-- Tabela Pessoa Fisica
CREATE SEQUENCE seq_TAB_Pessoa_Fisica INCREMENT 1 MINVALUE 1 MAXVALUE 9223372036854775807 START 1 CACHE 1;

-- Tabela Pessoa Fisica
CREATE TABLE TAB_Pessoa_Fisica(
        IDPessoaFisica INT NOT NULL DEFAULT NEXTVAL('seq_TAB_Pessoa_Fisica'),
        RG VARCHAR(8) NULL,
        CPF VARCHAR(11) NULL,
        Data_Nascimento Date NULL,
        IDPessoa INT NOT NULL,
        Primary key (IDPessoaFisica)
);

-- Foreign Key da tabela Pessoa Fisica

ALTER TABLE TAB_Pessoa_Fisica ADD FOREIGN KEY(IDPessoa) REFERENCES TAB_Pessoa (IDPessoa);
```

Fonte: Interna (2015).

Único método da classe **PessoaDao** que sofreu alteração de script foi o **Add** que precisou utilizar o ‘SELECT CURRVAL('seq_TAB_Pessoa')’ para retornar o ultimo valor utilizado da Sequencia **'seq_TAB_Pessoa'.** Segue como ficou o método.

Figura 30 – Método Add da classe PessoaDao.

```
/// <summary> Metodo que adiciona uma pessoa fisica ao banco de dados
public void Add(PessoaFisica pessoaFisica)
{
    string sql = string.Empty;
    NpgsqlCommand cmd = null;

    try
    {
        using (var conexao = new UnitOfWork.UnitOfWorkPG())
        {
            sql = "insert into TAB_Pessoa (nome) values (@nome); SELECT CURRVAL('seq_TAB_Pessoa');";

            cmd = new NpgsqlCommand(sql, conexao.Connection);
            cmd.Transaction = conexao.Transaction;
            cmd.Parameters.AddWithValue("@nome", pessoaFisica.Pessoa.Nome);
            pessoaFisica.Pessoa.IDPessoa = Convert.ToInt32((long)cmd.ExecuteScalar()); ;

            sql = string.Empty;
            sql = "insert into TAB_Pessoa_Fisica (RG, CPF, Data_Nascimento, IDPessoa) values(@rg, @cpf, @dataNascimento, @idPessoa)";

            cmd = new NpgsqlCommand(sql, conexao.Connection);
            cmd.Transaction = conexao.Transaction;
            cmd.Parameters.AddWithValue("@rg", pessoaFisica.RG);
            cmd.Parameters.AddWithValue("@cpf", pessoaFisica.CPF);
            cmd.Parameters.AddWithValue("@dataNascimento", pessoaFisica.DataNascimento);
            cmd.Parameters.AddWithValue("@idPessoa", pessoaFisica.Pessoa.IDPessoa);
            cmd.ExecuteNonQuery();

            conexao.Commit();
        }
    }
    catch (Exception)
    {
        throw;
    }
}
```

Fonte: Interna (2015).

8. CONCLUSÃO

Conclui-se, portanto, que desenvolver um software usando ORM se ganha produtividade, menos retrabalho e um código mais limpo. Para um sistema com muitas funcionalidades, muitas vezes detectar e corrigir problemas se tornou bastante complexo. Porém com o uso do ORM as consultas ficam ate três vezes mais rápidas. Diminui-se significativamente o numero de linhas de código de uma aplicação para outra. Com relação à parte de produtividade da aplicação com uso de ORM foi de grande auxilio, pois possibilitou que fosse implementado os requisitos com um prazo mínimo de tempo que não seria possível usando ADO Net. Mais a parte que mais ressalta e interdependência de banco de dados que com uma única linha de código alterada, migramos o banco de dados do sistema, sem que suas funcionalidades fossem afetadas.

9. REFERÊNCIAS

MICROSOFT. **ADO.NET Documentation**. Disponível em: < https://msdn.microsoft.com/en-us/library/e80y5yhx.aspx>. Acesso em: 03 de novembro 2015.

PATRICK, Tim. **Microsoft ADO.NET 4 Step by Step. 2010**: O'Reilly Media, Inc, 2010. 440 p.

PERKINS, Benjamin. **Working with NHibernate 3.0**: Wiley Publishing, Inc, 2011. 221 p.

LIPITSÄINEN, Arvo. **ORM – Object Relational Mapping**: DBTechNet Tutorial, 2011. 30 p.

GREGORY, James. **Fluent configuration.** Disponível em: <https://github.com/jagregory/fluent-nhibernate/wiki/Fluent-configuration>. Acesso em: 08 de novembro 2015.

MICROSOFT. **C# Documentation**. Disponível em: < https://msdn.microsoft.com/pt-br/library/ms228280(v=vs.90).aspx>. Acesso em: 03 de novembro 2015.