FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA” CENTRO UNIVERSITÁRIO “EURÍPIDES DE MARÍLIA” – UNIVEM BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DIEGO ROBERTO COLOMBO DIAS

AVALIAÇÃO DO USO DE SISTEMAS ORIENTADOS A OBJETOS UTILIZANDO BDOO (JAVA X .NET)

MARÍLIA
2008

DIEGO ROBERTO COLOMBO DIAS

AVALIAÇÃO DO USO DE SISTEMAS ORIENTADOS A OBJETOS UTILIZANDO BDOO (JAVA X .NET)

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação do Centro Universitário Eurípides de Marília (UNIVEM), mantido pela Fundação de Ensino Eurípides Soares da Rocha, para obtenção do Título de Bacharel em Ciência da Computação.

Orientador (a):
Profº. Drº Edmundo Sérgio Spoto.

MARÍLIA
2008

DIAS, Diego Roberto Colombo.

Avaliação do Uso de Sistemas Orientados a Objetos utilizando BDOO (Java x .NET) / Diego Roberto Colombo Dias / Edmundo Sérgio Spoto. Marília, SP: [s.n], 2008.

Apresentação de Monografia (Graduação em Ciência da Computação) – Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha.

1. Banco de Dados 2. Java 3. .NET

CDD: 005.74

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO – AVALIAÇÃO FINAL

Diego Roberto Colombo Dias

AVALIAÇÃO DO USO DE SISTEMAS ORIENTADOS A OBJETOS UTILIZANDO BDOO (JAVA X. NET)

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 8.0 (oito)

Orientador: Edmundo Sérgio Spoto

1°. Examinador: Paulo Augusto Nardi

2°. Examinador: Fabio Lucio Meira

Marília, 13 de novembro de 2008.

DIAS, Diego Roberto Colombo. Avaliação do uso de Sistemas Orientados a Objetos utilizando BDOO (Java x .NET). 2008.
Monografia (Bacharelado em Ciência da Computação) – Curso Ciência da Computação, Fundação de Ensino “Eurípides Soares da Rocha”, Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, Marília, 2008.

RESUMO

Com a necessidade de se abstrair objetos do mundo real na programação, surgiram as linguagens OO. Este trabalho apresenta a comparação entre desempenho de duas dessas linguagens, gerando resultados de avaliações efetuadas entre duas aplicações de plataformas distintas e um BDOO. Utilizando as plataformas Java, .NET e um BDOO construído em um SGBDOR, serão gerados vários casos de avaliação de desempenho, entre eles, inserção, recuperação, atualização e exclusão na base de dados. Serão também apresentadas todas as implementações das aplicações e do BDOO (métodos, polimorfismo, herança, e Objetos específicos). Ao final desse trabalho, por meio de gráficos e uma tabela comparativa, será mostrado qual das aplicações tem o melhor desempenho com o BDOO .

LISTA DE FIGURAS

LISTA DE ABREVEATURAS

BD: Banco de Dados

BDOO: Banco de Dados Orientado a Objetos

CAD: *Computer-Aided Design*

CASE: *Computer-Aided Software Engeneering*

CLR: *Common Language Runtime*

CLS: *Common Language Specification*

CTS: *Common Language System*

JBDC: *Java Data Base Connectivity*

JIT: *Just In Time*

JVM: *Java Virtual Machine*

MSIL: *Microsoft Intermediate Language*

MSMQ: *Microsoft Message Queuing*

ODBC: *Open Data Base Connectivity*

OID: O*bject Identifier*

OIS: *Office Information System*

OleDb: *Object Linking and Embedding Data Base*

OO: Orientado a Objetos

OQL: *Obejct Query Langage*

SGBD: Sistema Gerenciador de Banco de Dados

SGBDOO: Sistema Gerenciador de Banco de Dados Orientado a Objetos

SGBDOR: Sistema Gerenciador de Banco de Dados Objeto Relacional

SOA: *Service-Oriented Architecture*

SQL: *Structured Query Language*

WCF: *Windows Communication Foundation*

WCS: *Windows CardSpace*

WPF: *Windows Presentation Foundation*

WWF: *Windows Workflow Foundation*

WWW: *World Wide Web*

XAML: *Extensible Application Markup Language*

XML: *Extensible Markup Language*

LISTA DE GRÁFICOS

LISTA DE TABELAS

SUMÁRIO

INTRODUÇÃO

Com a necessidade de se abstrair o mundo real na programação, surgiram as linguagens OO. Desde então, várias linguagens foram criadas, para as mais diversas aplicações, *desktops* ou até dispositivos móveis. Entre elas temos as linguagens Java e a plataforma .NET, que serão abordadas neste trabalho.

Java é uma linguagem OO muito utilizada no mercado hoje em dia, com seu alto poder de produtividade, código pequeno e plataformas independentes, a linguagem tornou-se uma das mais utilizadas no mundo todo, para todos os tipos de aplicações.

.NET, em contrapartida, foi criada pela Microsoft como um pacote. Ele engloba vários *frameworks*, surgindo assim uma poderosa ferramenta de programação com alto poder de produtividade e de fácil adaptação, devido à aceitação de várias sintaxes diferentes.

Atualmente os bancos de dados são de extrema importância para todos os ramos de negócios. Eles são usados para manter registros internos, apresentar dados a consumidores e clientes na *World Wide Web* e fornecer suporte a vários outros processos comerciais, também são encontrados no núcleo de muitas investigações científicas, em dados reunidos por astrônomos, por investigadores do genoma humano e por bioquímicos que exploram as propriedades medicinais (Windom, Garcia-Molina et al., 2001).

O desígnio dos sistemas de banco de dados é o gerenciamento de grandes quantidades de informações e arquivos. Esses sistemas evoluíram primeiro em bancos de dados hierárquicos ou em rede, e depois em bancos de dados relacionais (Korth, Silberschatz et al., 1999).

O Banco de Dados Relacional não passa de um repositório de dados compartilhado. Como o próprio Silberschatz afirma, “muitas áreas de aplicação para sistemas de banco de dados são limitadas por restrições do modelo de dados relacional” (Korth, Silberschatz et al., 1999).

Os programadores OO precisavam de mais do que um simples repositório de dados, assim surge à necessidade de se criar um novo conceito de armazenamento de dados. Dando origem aos primeiros BDOO’s.

Segundo *The Object-Oriented Database Manifesto*, Malcolm Atkinson e outros definem um SGBDOO como: “um sistema gerenciador de Banco de Dados Orientado a Objetos, é aquele que suporta a modelagem e criação de dados como objetos”. Dessa maneira surgiu a idéia de que os SGBDOO’s seriam criados para substituir os SGBD’s relacionais, pois eles trabalham de forma mais simples com as linguagens OO. Entretanto,

isso não aconteceu devido aos altos custos de migração dos dados. Dessa maneira surgiram os SGBDOR, que são formas de mapear os objetos em bancos de dados relacionais.

O surgimento do BDOO gerou a dúvida de qual linguagem OO se usar com determinado SGBD (OR ou OO). Isso gerou a necessidade de se realizar avaliações para saber qual linguagem tem melhor desempenho com os objetos na base de dados.

Um SGBD é uma reunião de programas que possibilita ao usuário criar e manter um banco de dados. Portanto, o SGBD é um sistema de software de propósito geral que facilita os processos de definição, construção, manipulação e compartilhamento de bancos de dados entre vários usuários e aplicações (Elmasri e Navathe, 2005).

Objetivo

O objetivo deste trabalho é apresentar um estudo comparativo de desempenho de um Banco de Dados Orientado a Objetos entre duas aplicações iguais, porém, programadas em linguagens diferentes: Java e .NET.

A partir das aplicações devem ser feitas inserção, recuperação, atualização e exclusão em uma base de dados Oracle9i desenvolvida com técnicas OO, assim obtendo os tempos de execução de cada instrução SQL executada no BD pelas duas aplicações.

Em relação ao BDOO, pretende-se também testar a utilização de métodos para várias quantidades de elementos cadastrados nas tabelas de objetos.

Uma análise de resultados deverá ser realizada visando comparar o comportamento dos métodos exercitados diante de grandes quantidades de objetos armazenados.

Organização do Trabalho

Este trabalho de conclusão possui além desta introdução, três capítulos, a saber:

- No **capítulo 1** são apresentados históricos de conceitos de todos os temas abordados neste trabalho: BDOO, Java e .NET.
- No **capítulo 2** são apresentados detalhes das duas aplicações propostas para os testes. Também são apresentadas as formas de avaliações que serão aplicadas neste trabalho.

- No **capítulo 3** é detalhada a execução das avaliações. Também são apresentados os resultados obtidos e análise desses resultados. Por último, são apresentados os problemas a serem resolvidos.

Finalizando, tem-se a conclusão, a definição de trabalhos futuros e as referências que serviram como embasamento teórico para a elaboração deste trabalho de conclusão.

CAPÍTULO 1 – CONCEITOS BÁSICOS E TERMINOLOGIA

1.1 Banco de Dados Orientado a Objetos

Com o surgimento do paradigma OO houve um avanço na área de reuso de código em linguagens de programação. Com isso a Orientação a Objetos torna uma área aliada e desejável aos projetos de Banco de Dados com as características do paradigma OO. O que difere do conceito de BD Relacional é que a OO proporciona criações de métodos além dos atributos, contribuindo com várias características positivas em relação ao projeto de Banco de Dados Orientado a Objetos (Booch, Jacobson, Coleman e Rambaught, 2006).

1.1.1 O que é BDOO

Os primeiros BDOO`s surgiram no final da década de 80, para suprir a necessidade dos programadores OO em armazenar dados mais complexos, como: banco de dados multimídia, *OIS* (*Office Information System* – banco de dados que permitem consultas a horários, documentos e conteúdo de documentos na automação de escritórios), *CAD* (*Computer-Aided Design* – banco de dados que armazena dados de projetos de engenharia), *CASE* (*Computer-Aided Software Engeneering* – banco de dados usado para armazenar dados requeridos para auxiliar desenvolvedores de software), sistemas de banco de dados especialistas (Korth, Silberschatz et al., 1999).

O BDOO segue o mesmo paradigma da programação orientada a objetos, usando conceitos de dados encapsulados e acessados através de seus métodos (Korth, Silberschatz et al., 1999).

Existem vários sistemas disponíveis comercialmente, entre eles o GEMSTONE/OPAL, da GemStone System; ONTOS, da Ontos; ARDENT, da Ardent Software. Esses sistemas apesar de serem SGBDOO, não são muito utilizados, pois é preciso usar uma linguagem padrão. Com isso o SGBD objeto relacional ainda tem grande parte no mercado, uma vez que utilizam padrões de linguagem comuns (Elmasri e Navathe, 2005).

O surgimento do BDOO fez com que os fornecedores de SGDBs relacionais adicionassem características OO em seus produtos, surgindo assim o Objeto Relacional.

1.1.2 Conceitos BDOO

- **Identidade de Objeto**

Um objeto, como na programação orientada a objetos, é a forma de se abstrair um objeto do mundo real para o banco de dados OO. Porém no BDOO cada objeto armazenado no banco de dados tem uma identidade única. Esta identidade é um identificador de objeto ou um OID (Elmasri e Navathe, 2005).

O OID é invisível ao usuário, mas é utilizado sempre que o sistema queira referenciar ou identificar um objeto armazenado no banco de dados OO. O OID é responsável por saber com qual objeto se comunicar no banco de dados (Elmasri e Navathe, 2005).

O OID tem duas importantes propriedades: o seu valor não deve ser modificado, e mesmo que o objeto não exista mais no banco de dados, o OID dele não deve ser usado novamente (Elmasri e Navathe, 2005).

- **Tipos e Classes**

Uma classe define o valor dos dados armazenados e a funcionalidade associada com o objeto daquela classe. Cada objeto pertence somente a uma classe, isto é, um objeto é instanciado a partir de uma classe. Em um SGBDOR a classe construtora é usada para definir o esquema do BD. Alguns SGBDOR's usam o termo tipo para definir classes.

Na Figura 1 é criado um objeto no Banco de Dados, do tipo Funcionário com os atributos: *codFuncionario*, *cargo*, *nomeFunc*, *dataAdmissao*, *dataNasc*, *salário*, *setor*, *endereoFunc* e *telefoneFunc*.

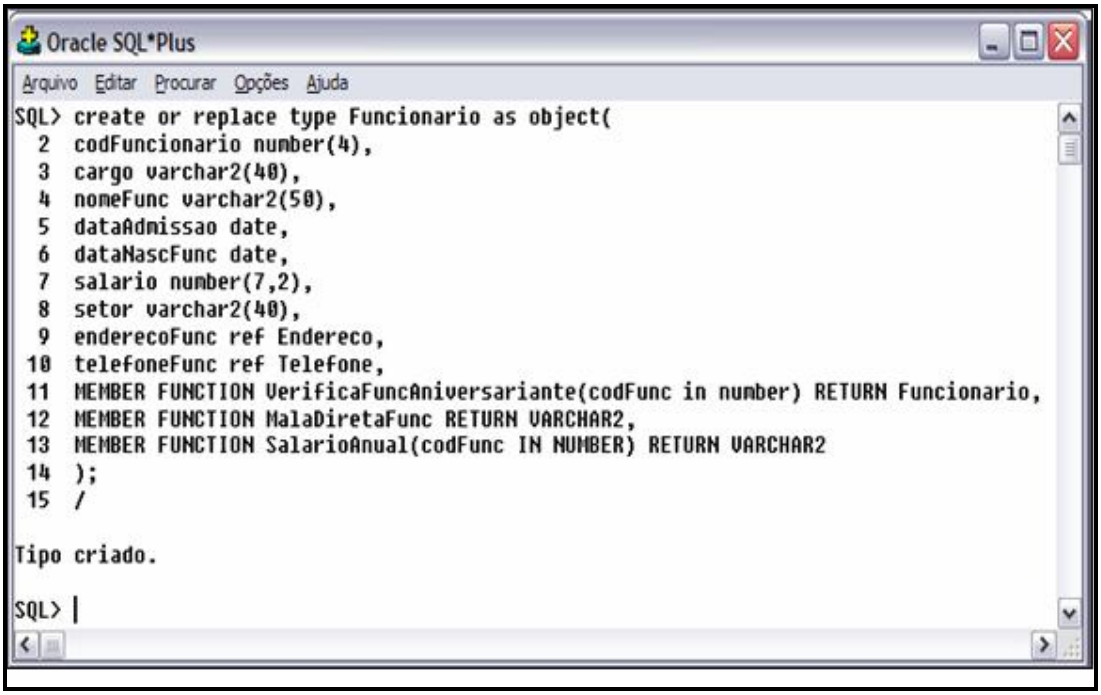


Figura 1 – Tipos e Classes (Fonte: Ruiz e Gonçalves, 2007).

Os atributos representam os dados que fazem parte do conteúdo de uma classe. Cada objeto instanciado tem seus próprios atributos armazenados.

Os tipos de atributos são basicamente subconjuntos de tipos básicos suportados por linguagens de programação que interagem com o SGBD. Esses tipos incluem tipos inteiros, *floats*, *strings*, *char*, entre outros.

- **Encapsulamento**

O encapsulamento consiste em esconder do usuário a implementação interna e o comportamento das classes. Isto permite que o usuário não seja afetado quando a implementação é modificada, provendo assim uma independência dos dados. Os atributos de uma classe podem ou não ser encapsulados. Se a classe não for encapsulada, os seus atributos também não serão.

- **Herança**

A herança permite que uma classe incorpore os atributos e comportamentos de uma ou mais classes. Essa classe então é chamada de subclasse. A subclasse é uma especialização de uma superclasse, isto é, além de incorporar tudo da superclasse, ainda pode adicionar dados ou comportamentos próprios.

Na modelação OO, a herança é um conceito poderoso, pois ela suporta o reuso e a extensibilidade de classes existentes.

O relacionamento de herança entre um grupo de classes, define uma hierarquia dessas classes, ajudando assim que o usuário entenda o conhecimento de uma classe (superclasse) para ser aplicada a outra classe (subclasse).

Um exemplo de herança do Banco de Dados usado nesse estudo é mostrado na Figura 2. É definido um tipo Cliente, que é a superclasse das outras duas classes clienteFísico e clienteJurídico.

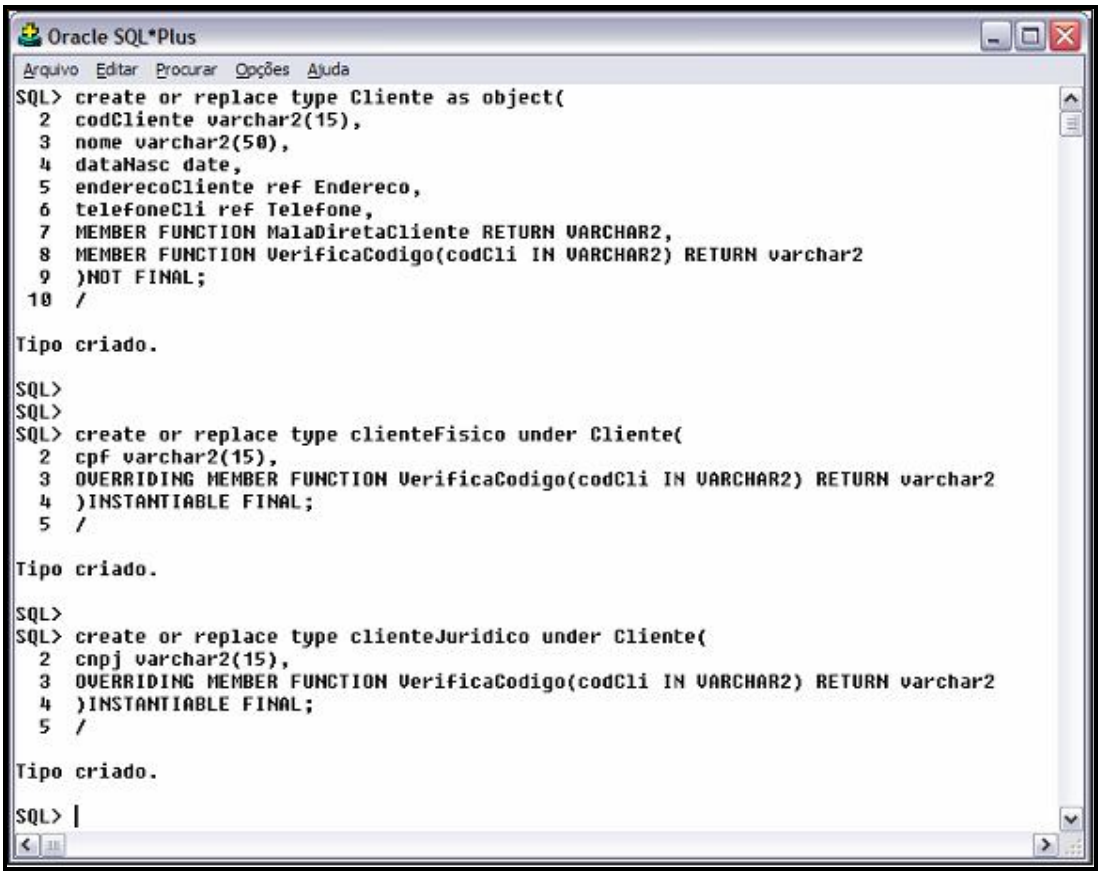


Figura 2 – Herança (Fonte: Ruiz e Gonçalves, 2007).

- **Persistência**

Um SGBDOR é responsável por fazer objetos persistentes. A persistência de um objeto é normalmente definida quando o objeto é criado. Um objeto persistente é aquele que é armazenado no banco de dados depois da execução do programa.

Para que um objeto persistente seja encontrado em banco de dados, ele deve receber um nome que é dado através de um mecanismo de nomeação, isto é, o objeto recebe um nome único pelo qual ele possa ser encontrado no banco de dados.

Mas esse mecanismo não é utilizado para todos os objetos de um banco de dados, na maioria deles é utilizado um outro mecanismo, a alcançabilidade. Ela faz com que um objeto que esteja ligado a um objeto persistente, seja encontrado pelo grafo de ligação deles.

O objeto persistente tem referências a outros objetos, isto é, ele pode ser removido somente quando todas suas referências tiverem sido removidas.

1.1.3 Sistema Gerenciador de Banco de Dados Objeto Relacional

Um SGBDOR suporta a modelagem e criação de dados como objetos. Isso implica em classes e métodos para se trabalhar com esses objetos. Garcia-Molina definiu como tendo dois critérios para que o sistema seja um SGBDOR: deve ser um SGBD e também deve ser orientado a objetos e relacional.

Um SGBD é uma ferramenta poderosa para criar e gerenciar grandes quantidades de dados de forma eficaz, e permitir que esses dados persistam durante logos espaços de tempos com segurança (Windom, Garcia-Molina et al., 2001).

Isso é, para satisfazer o critério de ser SGBD, ele deve ter:

- Persistência,
- Controle de armazenamento secundário,
- Concorrência,
- Recuperação
- *Ad hoc query*.

O modelo OO é baseado no paradigma da programação orientada a objeto (Korth, Silberschatz et al., 1999). Assim, o critério OO deve ser capaz de implementar:

- Objetos complexos,
- Identidade de um objeto (OID),
- Encapsulamento,
- Tipos ou classes,
- Herança,
- Sobrecarga de métodos,
- Extensibilidade
- Completude computacional.

Por não haver uma linguagem padrão para se trabalhar com SGBDOO, como o SQL para os SGBD`s, e por existir um alto custo para se migrar o sistema, foi criado um novo padrão, o SGBDOR (Sistema Gerenciador de Banco de Dados Objeto Relacional), que nada mais é do que um SGBD com mapeamentos OO, sendo interessante para quem já tem uma Base de Dados Relacional e queira migrar para OO.

Os BDOO's foram estabelecidos como um complemento, e não como uma substituição aos BDR (Banco de Dados Relacionais).

Em 2001 a empresa *Object Data Management* começou uma padronização *Object Query Language* (OQL), contudo o projeto foi abandonado (Windom, Garcia-Molina et al., 2001).

Desta forma utilizaremos neste trabalho um SGBDOR, visando aplicar as regras de um BDOO acessado por linguagens Java e .Net.

1.2 Java

1.2.1 O que é Java

A linguagem Java inicialmente foi criada para ser usada em equipamentos de consumo como caixas de comutação de TV a cabo. Como esses dispositivos não tinham grande poder de processamento ou memória de armazenamento, a linguagem tinha que gerar um código bem enxuto.

Com isso, os projetistas de um projeto chamado "GREEN", um grupo de projetistas da Sun Microsystem, usou como modelo a idéia de uma linguagem chamada UCSD Pascal, uma linguagem mais antiga criada por Niklaus Wirth. A idéia era usar uma linguagem

portável que gerasse um código intermediário para uma máquina virtual interpretá-la, podendo assim esse código ser interpretado por qualquer dispositivo que tivesse o interpretador correto. Os códigos intermediários que são gerados são sempre pequenos e seus interpretadores também.

O projeto “GREEN”, em 1992, apresentou um produto chamado “*7”, que era um controle remoto inteligente. Mas mesmo assim, a Sun não estava interessada em produzir esse dispositivo, como nenhuma outra empresa. Por isso, o grupo que passou a se chamar First Person Inc, começou a procurar novas maneiras de comercializar sua tecnologia. Eles ofereceram um projeto de uma caixa de TV a cabo que funcionaria com novos serviços a cabo, mas não conseguiram fechar nenhum contrato (Cornell e Horstmann, 2004).

Eles passaram cerca de dois anos tentando conseguir alguém que comprasse sua tecnologia, não obtendo sucesso. Até que em 1994, a empresa foi dissolvida (Cornell e Horstmann, 2004).

Já a Sun Microsystem utilizou como idéia principal a linguagem C++ à vez da Pascal. Com isso, eles criaram uma linguagem OO ao invés de uma linguagem procedimental.

Segundo Gosling (Gosling, 2004): “Desde o início, a linguagem era uma ferramenta, não o fim”. Ele então deu o nome ao projeto de “OAK”. Muitos acham que ele deu esse nome por gostar muito da aparência de um carvalho que ficava a frente da sua janela na Sun. Porém mais tarde descobriram que já havia uma linguagem com esse nome, passando-se então a se chamar Java.

A Sun estava aproveitando a parte WWW da internet, que estava crescendo cada vez mais, criando vários navegadores para internet, entre eles o HotJava, que era um navegador escrito totalmente em Java, mostrando assim o poder da linguagem.

Porém, hoje a linguagem não é usada só para esses tipos de aplicações descritos acima, é também uma linguagem poderosa como ferramenta de programação, podendo ser utilizada para aplicações em geral, como multimídia, interfaces gráficas e banco de dados (Cornell e Horstmann, 2004).

1.2.2 Conceitos Básicos Java

A linguagem Java possui alguns conceitos muito úteis, que faz com que ela seja uma boa escolha no desenvolvimento de aplicações em geral.

Distribuída: o Java foi desenvolvido para desenvolver aplicações distribuídas, o que consiste em programas residentes em computadores diferentes numa rede que cooperam uns com os outros.

Plataformas Independentes: um ponto forte da plataforma Java, é sua execução em plataformas diferentes. Isto faz com que o Java seja a escolha perfeita na hora de lidar com heterogeneidade numa rede de computadores, como na internet, eliminando assim a preocupação em ter que portar seus programas cada vez que se tenha, por exemplo, um Sistema Operacional diferente (Jia, 2000).

- **Java Byte-Code e JVM**

O Java possui uma JVM (*Java Virtual Machine*) que é responsável pela sua compilação. Uma JVM reconhece *byte-codes.*

Os códigos Java podem ser executados de três maneiras diferentes:

- *Interpretação*: uma plataforma interpreta as instruções da JVM e as executa. Essa é a forma mais original e suportada pela Sun.
- *Compilação* JIT *(Just In Time)*: uma plataforma e mais um compilador *just-in-time*, compilam o *byte-code* para código de máquina nativo e executa esse código.
- *Execução Direta*: são chips com a JVM embutida que pode assim ser executada diretamente sem um compilador ou interpretador (Jia, 2000).

1.3 .NET

1.3.1 O que é .NET

A plataforma .NET é um conjunto de cinco *frameworks* que foi construída pela Microsoft para agilizar o processo de desenvolvimento de aplicações. O projeto de

desenvolvimento do *framework* foi iniciado em julho de 2000, e a primeira versão foi lançada em abril de 2003, e a última é chamada de .NET 3.5 (Lam e Thai, 2003).

A plataforma .NET 3.5 tem como componentes o .NET 2.0, que é responsável pelas APIs, tipos, coleções, *networking*, segurança, *threads*, aplicativos *web* e aplicativos Windows. Já entre os outros componentes, estão novas tecnologias poderosas para facilitar o processo de desenvolvimento de software (MSDN 1, 2008)..

Os quatros novos *frameworks* que fazem parte do .NET 3.5 são: *Windows Presentation Foundation* (WPF), *Windows Communication Foundation* (WCF), *Windows Workflow Foundation* (WWF) e *Windows CardSpace* (WCS). Na Figura 3 são apresentados os componentes do *framework* .NET 3.5 (MSDN 1, 2008)..

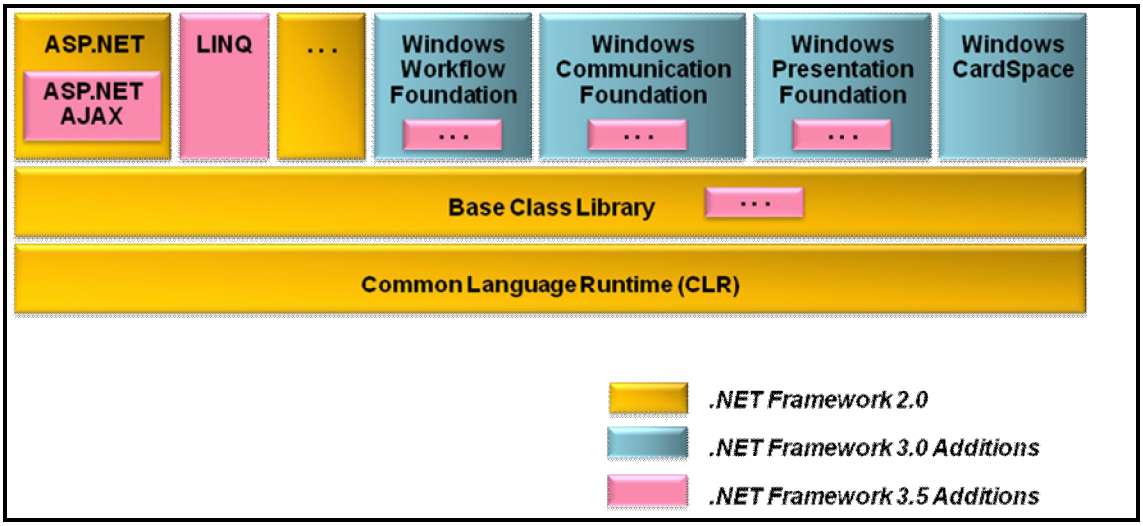


Figura 3 – Componentes .NET 3.5 (Fonte: Introducing the .NET Framework 3.5)

O *Windows Presentantion Foundation* (WPF) possui as bibliotecas responsáveis pela parte gráfica 3d, 2d e *layouts*. Por ser uma tecnologia criada pela Microsoft, o WPF utiliza como auxílio o *Direct X*, responsável pelos recursos gráficos oferecidos por *hardware*. Ele ainda utiliza um novo modelo de linguagem derivada do XML, linguagem essa denominada de XAML, que foi criada para se construir interfaces gráficas. Ele agrupa as varias tecnologias de conexão utilizadas pela Microsoft, tais como *Web Service*, *Web Remoting*, MSMQ e *Enterprise Services* (MSDN 2, 2008).

O *Windows Communication Foundation* (WCF) é baseado no modelo de programação que vem sendo muito utilizado ultimamente, o *Service-Oriented Architecture* (SOA). O WCF, então, é uma instância de SOA, pois implementa os seus conceitos (MSDN 3, 2008)..

O *Windows Workflow Foundation* (WWF) é uma plataforma de desenvolvimento de *workflows* (conjunto de atividades ordenadas). A utilização dessa ferramenta fornece um modelo para que o desenvolvedor projete, desenhe e execute seus códigos. Os aplicativos desenvolvidos podem comunicar com WF (MSDN 4, 2008)..

O *Windows CardSpace* (WCS) é um *framework* destinado a segurança, tendo como principal foco as identidades digitais. No mundo real temos varias identidades, como RG, passaporte ou um cartão de créditos, isto é, temos várias identidades diferentes, mas que se referem à mesma pessoa. Também foi criado para que qualquer aplicativo Windows tenha um modo comum para tratar as identidades digitais (MSDN 5, 2008).

1.3.2 Conceitos Básicos .NET

- ***Common Language Runtine* (CLR) e *Common Type System* (CTS)**

O CLR é a base da plataforma .NET. É uma camada de *software* carregada pelo sistema operacional que verifica se as aplicações podem ser executadas sem erros, define as permissões de segurança e libera os recursos necessários (Lam e Thai, 2003).

As aplicações carregadas pelo *CLR* são escritas em qualquer das linguagens suportadas pelo .NET. O código fonte depois de compilado, gera um código chamado *Microsoft Intermediate Language* (*MSIL*), conhecido também como *managed code*. Esse código é carregado, traduzido para código de máquina e monitorado em tempo de execução (Filho et al, 2002).

Segundo Santana (Filho et al, 2002): "*O objetivo do CLR é aumentar a confiabilidade e segurança em aplicações e reduzir o volume de repetição de código – propício a erros -, além de facilitar o acesso a recursos específicos do sistema operacional*".

As linguagens suportadas pela plataforma .NET devem emitir metadados, esses que são emitidos pelo compilador da linguagem. Os metadados são informações que descrevem os tipos, funções e referências do código.

O CLR recebe o *manage code* em um ou vários arquivos chamado *assembly*. Esse contendo o código MSIL, os metadados e um *manifest*, que é um documento responsável pelo relacionamento de todos os arquivos e componentes no *assembly*. (Filho et al, 2002).

Segundo Santana (Filho et al, 2002), os metadados são utilizados para localizar e carregar classes, formulários na memória, resolver chamadas de métodos, gerar código nativo e aumentar a segurança.

O CLR também cuida para que componentes gerados por diferentes linguagens possam interagir. Isto é possível devido ao CTS (*Common Type System*), um sistema comum de tipos, para que os componentes sejam compatíveis. O CTS permite que classes implementadas em linguagens diferentes sejam integradas, através de regras estabelecidas. (Lam e Thai, 2003).

- ***Class Library***

A *Class Library* é um conjunto de classes da plataforma .NET utilizada para desenvolvimento de aplicativos diversos. Ela é composta de classes, interfaces e tipos otimizados para o processo de desenvolvimento, podendo ser usada por qualquer linguagem suportada pela plataforma, por estar de acordo com a *Common Language Specification* (CLS) (Filho et al, 2002).

1.4 Conceitos OO

- **Objeto e Classe**

Os objetos e classes são dois conceitos fundamentais em Java e .NET. Eles representam os aspectos de representação em um modelo orientado a objetos e a interpretação do mundo real.

Todo objeto possui um estado, que é representado pelos seus campos ou atributos e seus valores. O estado do objeto é alterado pelos seus métodos.

Quando se tem vários objetos similares, isto é, com atributos e métodos em comum, cria-se uma classe. A partir dessa classe os objetos são instanciados. A classe caracteriza a estrutura dos estados e comportamentos que são compartilhados por todas suas instâncias. Na Figura 4 temos um exemplo de uma classe com seus atributos e um método construtor, utilizado para inicializar a classe (Sierra e Bates, 2007).

```
public class Endereco {//Nome da classe
    int codEnd;
    String cep;
    String cidade;
    String estado;
    String rua;

    public Endereco() {//Construtor da classe Endereco

    }
}
```

Figura 4 – Classe

- **Abstração e Encapsulamento**

A abstração e o encapsulamento são ferramentas poderosas para modularizar e decompor o sistema.

A abstração cuida para que sejam separadas as características essenciais das não essenciais de uma entidade. Dessa maneira o comportamento ou funcionalidades de um módulo devem ter uma descrição precisa conhecida como *interfaces contratuais*. Com isso, as interfaces contratuais capturam a essência do comportamento do módulo.

O encapsulamento é responsável por reduzir o acoplamento entre os módulos. Em outras palavras isso quer dizer que, quanto menos o usuário conhecer da implementação de um módulo, menor será o acoplamento entre o usuário e o módulo. Isso é importante para que o usuário não saiba como o módulo foi implementado, podendo assim ser modificado sem afetá-lo (Sierra e Bates, 2007)..

- **Herança**

A herança na programação OO ocorre quando uma classe herda as propriedades de uma superclasse, podendo adicionar suas próprias características.

Na Figura 5 temos um exemplo de definição de uma **superclasse** do Banco de Dados utilizado nesse trabalho. Uma classe *Cliente* é definida como **superclasse** (Sierra e Bates, 2007).

```
 */
public class Cliente {

    String codCliente;
    String nome;
    Date dataNasc;

    public Cliente() {

    }
}
```

Figura 5 – Classe Cliente

As Figuras 6 e 7, são classes *estendidas* da Figura 5. Isto é, as duas classes herdam todos os atributos e métodos da superclasse *Cliente* e modificam ou adicionam os seus próprios métodos.

```
public class clienteFísico extends Cliente {

    String cpf;

    public clienteFísico() {

    }
}
```

Figura 6 – Classes clienteFísico

```
public class clienteJurídico extends Cliente {

    String cnpj;

    public clienteJurídico() {

    }
}
```

Figura 7 – Classe clienteJurídico

- **Polimorfismo**

O Polimorfismo é a habilidade de um objeto mudar seu comportamento dinamicamente, sem afetar o usuário. Essa mudança é feita em tempo de execução. O polimorfismo pode ser aplicado a qualquer objeto que seja herdado de uma superclasse.

O uso do polimorfismo na programação OO possibilita a declaração de variáveis de referência, utilizando a superclasse como referência e suas subclasses como objeto, podendo assim ser criadas matrizes polimórficas (Sierra e Bates, 2007).

Na Figura 8 temos um exemplo de matriz polimórfica utilizando a superclasse Cliente e suas subclasses, clienteFísico e clienteJurídico:

```
public class Main {

    Cliente[] clientes = new Cliente[5];

    public void Matriz() {
        clientes[0] = new clienteFisico();
        clientes[1] = new clienteJuridico();

    }

    public static void main(String[] args) {
    // TODO code application logic here
    }
}
```

Figura 8 – Matriz Polimórfica

- **Sobreposição e Sobrecarga de Métodos**

A sobreposição ocorre quando uma subclasse precisa modificar um ou mais métodos herdados da superclasse. Na sobreposição os argumentos e retornos devem continuar exatamente como o método sobreposto da superclasse. Nesse caso é utilizado o conceito de polimorfismo, pois se tem métodos idênticos a visão do compilador com os mesmos argumentos e retornos, porém o compilador sabe qual usar quando o método for solicitado (Sierra e Bates, 2007).

A sobrecarga acontece quando há a necessidade de se ter dois métodos com o mesmo nome, porém com argumentos diferentes. Nesse caso, diferente da sobreposição, não há polimorfismo, pois apesar de os métodos terem o mesmo nome, para o compilador eles são diferentes (Sierra e Bates, 2007).

1.5 Avaliação de Desempenho

O desempenho de um *software* esta relacionado ao tempo de execução de uma tarefa que ele possa realizar para um conjunto de valores que serão considerados como base para o cálculo do tempo de execução.

Para que ocorra uma avaliação de desempenho, devem se ter sempre dois elementos a serem comparados, sendo eles *software* ou *hardware*. Porém nem sempre essa combinação necessita ser utilizada.

Resultados obtidos em avaliações são comparados entre si, apresentando assim o elemento com melhor desempenho dentre os elementos avaliados.

Um exemplo da aplicação de avaliação de desempenho referente a software, é o trabalho de Mauro Sérgio Ribeiro (SOUZA, 1998), o qual utiliza a avaliação de desempenho para a mineração de dados em um banco de dados, por meio de algoritmos implementados pelo próprio autor.

Müller (2006) propõe várias métricas que avaliam a eficiência e a qualidade de um *software* sob a Norma da ISO IEC 9126 (2006). Que classifica o bom desempenho para garantir uma ótima qualidade de um produto de software e sua eficiência nas atividades funcionais e não funcionais.

Delfino (2004) faz o uso de avaliações em SGBD gratuitos, usados no gerenciamento de imagens médicas, estudando formas de armazenamento e recuperação dessas imagens.

Assim vários autores classificam a importância de avaliar um bom desempenho de um *software* para que possa no futuro aprender a melhorar a aplicabilidade em suas características funcionais, sendo elas para acesso a banco de dados ou simplesmente para acesso pelos usuários em suas funcionalidades.

CAPÍTULO 2 – GERAÇÃO DAS APLICAÇÕES OO

As ferramentas utilizadas para o desenvolvimento das aplicações foram:

- **IDE NetBeans 6.0.1** para o desenvolvimento da aplicação Java (NetBeans, 2008);
- **Microsoft Visual J# 2005 Express Edition** para o desenvolvimento da aplicação .NET (Microsoft, 2005);
- **Oracle 9i** SGBD utilizado para a execução do *Script* OO a ser utilizado nas avaliações (Oracle, 2005).

2.1 Aplicação Java

Para o desenvolvimento da aplicação *Java*, três classes foram criadas: ***ConexãoDB***, ***StopWatch*** e ***Interface***.

A classe ***ConexãoDB*** é responsável por todos os métodos relacionados à conexão com a base de dados Oracle9i e com os métodos de execução SQL. Os métodos são:

- *Conectar ( )*, que se encarrega de registrar o *driver* do banco de dados Oracle9i;
- *getConexao (String bd, String user, String pass)*, que é responsável por pegar os parâmetros de conexão do banco de dados, o nome do banco de dados, o usuário do banco e a senha para acesso ao banco. Parâmetros esses passados pela classe *Interface*;
- *CloseConexão ( )*, responsável por finalizar a conexão com o banco de dados;
- *insertSql (Boolean autocomm, Integer vezes, String sql)*, responsável pela execução da SQL utilizando o modo *PreparedStatement*, por meio dos parâmetros: *autocomm* (parâmetro que especifica se o *INSERT* será executado com *autocommit* desativado), *vezes* (parâmetro que especifica o número de vezes que será executada a instrução), *sql* (parâmetro que especifica a SQL que será executada). Parâmetros esses passados pela classe *Interface*;
- *insertSqlStat (Boolean autocomm, Integer vezes, String sql)*, responsável pela execução da SQL utilizando o modo *Statement*, por meio dos parâmetros: *autocomm* (parâmetro que especifica se a inserção será executada com *autocommit* desativado), *vezes* (parâmetro que especifica o número de vezes

que será executada a instrução), *sql* (parâmetro que especifica a SQL que será executada). Parâmetros esses passados pela classe *Interface*;

- *execincrSql (Boolean autocomm, Integer vezes, String sql),* responsável pela execução das SQL's utilizando o modo de execução *Prepared Statement*, por meio dos parâmetros: *autocomm* (parâmetro que especifica se a inserção, atualização ou exclusão serão executados com *autocommit* desativado), *vezes* (parâmetro que especifica o número de vezes que será executada a instrução), *sql* (parâmetro que especifica a SQL que será executada). Parâmetros esses passados pela classe *Interface*;
- *execincrSqlStat (Boolean autocomm, Integer vezes, String sql),* responsável pela execução das SQL's utilizando o modo de execução *Statement*, por meio dos parâmetros: *autocomm* (parâmetro que especifica se a inserção, atualização ou exclusão serão executados com *autocommit* desativado), *vezes* (parâmetro que especifica o número de vezes que será executada a instrução), *sql* (parâmetro que especifica a SQL que será executada). Parâmetros esses passados pela classe *Interface*;
- *selectSql (Integer vezes, String sql),* responsável pela recuperação na base de dados utilizando o modo *Prepared Statement,* por meio dos parâmetros: *vezes* (parâmetro que especifica o número de vezes que será executada a instrução), *sql* (parâmetro que especifica a SQL que será executada). Parâmetros esses passados pela classe *Interface*;
- *selectSqlStat (Integer vezes, String sql),* responsável pela recuperação na base de dados utilizando o modo *Statement,* por meio dos parâmetros: *vezes* (parâmetro que especifica o número de vezes que será executada a instrução), *sql* (parâmetro que especifica a SQL que será executada). Parâmetros esses passados pela classe *Interface*;

A classe ***StopWatch*** é responsável por todos os métodos relacionados a cronômetro. Os métodos são:

- *Start ( ),* responsável por pegar o tempo corrente que será o tempo inicial;
- *Stop ( ),* responsável por pegar o tempo corrente que será o tempo de parada;
- *getElapsedTime ( ),* responsável por fazer a subtração do tempo de parada pelo tempo inicial, retornando o tempo de execução em milisegundos;

- *getElapsedTimeSecs ( )*, responsável por fazer a subtração do tempo de parada pelo tempo inicial, retornando o tempo de execução de segundos.

A classe Interface é responsável por todos os métodos relacionados à interação com o avaliador (pessoa que realiza as execuções SQL). Ela implementa os métodos ***get*** e ***set*** de todos os campos declarados no *JFrame* (janela onde os campos são mostrados para o testador) e também as atividades dos botões, responsáveis pela ativação dos métodos. Na Figura 10 é apresentado um exemplo da interface gráfica da aplicação Java.

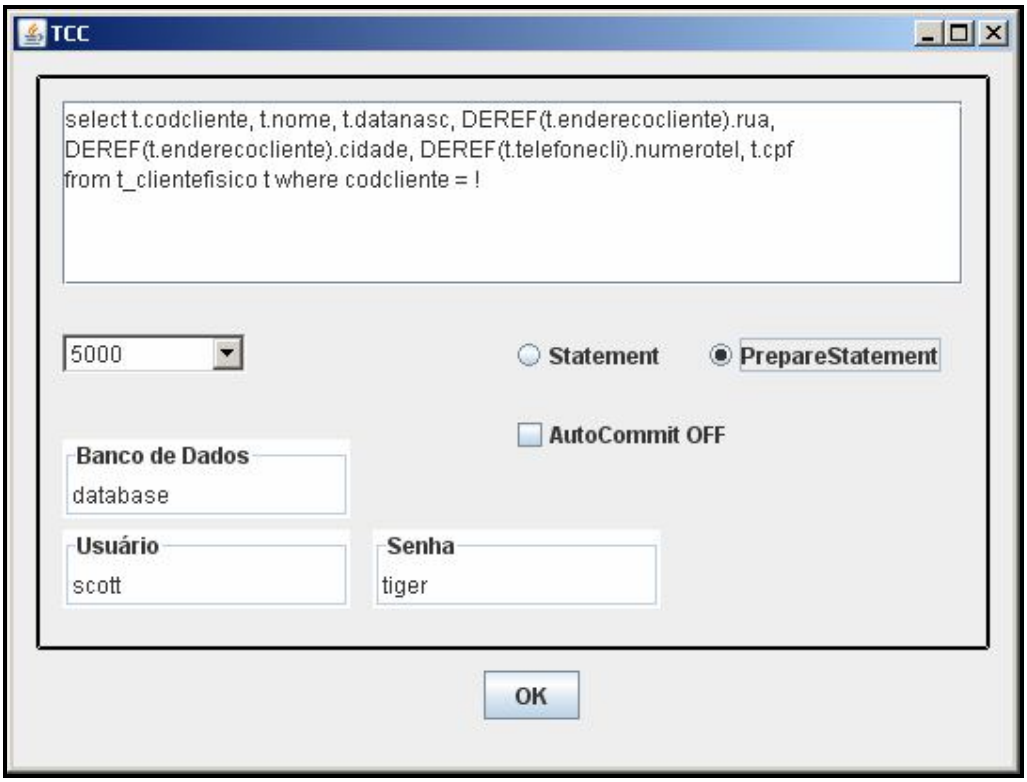


Figura 9 – Interface Gráfica da Aplicação Java

2.2 Aplicação .NET

Para o desenvolvimento da aplicação .NET, foram criadas duas classes: ***ConexãoDB*** e ***Interface***.

A classe ***ConexãoDB*** é responsável por todos os métodos relacionados a conexão com a base de dados Oracle9i e métodos de execução SQL. Os métodos são:

- *getConexao (String bd, String user, String pass)*, responsável por pegar os parâmetros de conexão do banco de dados, o nome do banco de dados, o usuário do banco e a senha para acesso ao banco. Parâmetros esses passados pela classe *Interface*;
- *CloseConexao ( )*, responsável por finalizar a conexão com o banco de dados;
- *insertSqlODBC (Boolean autocomm, Integer vezes, String sql)*, responsável pela execução da SQL usando o modo ODBC, por meio dos parâmetros: *autocomm* (parâmetro que especifica se o *INSERT* será executado com *autocommit* ou não), *vezes* (parâmetro que especifica o número de vezes que será executada a instrução), *sql* (parâmetro que especifica a SQL que será executada). Parâmetros esses passados pela classe *Interface*;
- *insertSqlOleDb (boolean autocommit, String vezes, String sql)*, responsável pela execução da SQL usando o modo *OleDbConection*, por meio dos parâmetros: *autocomm* (parâmetro que especifica se o *INSERT* será executado com *autocommit* ou não), *vezes* (parâmetro que especifica o número de vezes que será executada a instrução), *sql* (parâmetro que especifica a SQL que será executada). Parâmetros esses passados pela classe *Interface*;
- *execincrSqlODBC (Boolean autocomm, Integer vezes, String sql)*, responsável pela execução das SQL's utilizando o modo de execução ODBC, por meio dos parâmetros: *autocomm* (parâmetro que especifica se a inserção, atualização ou exclusão serão executados com *autocommit* desativado), *vezes* (parâmetro que especifica o número de vezes que será executada a instrução), *sql* (parâmetro que especifica a SQL que será executada). Parâmetros esses passados pela classe *Interface*;

- *execincrSqlOleDb (Boolean autocomm, Integer vezes, String sql),* responsável pela execução das SQL's utilizando o modo de execução OleDb, por meio dos parâmetros: *autocomm* (parâmetro que especifica se a inserção, atualização ou exclusão serão executados com *autocommit* desativado), *vezes* (parâmetro que especifica o número de vezes que será executada a instrução), *sql* (parâmetro que especifica a SQL que será executada). Parâmetros esses passados pela classe *Interface*;
- *selectSql (Integer vezes, String sql),* responsável pela recuperação na base de dados usando o modo *ODBC,* por meio dos parâmetros: *vezes* (parâmetro que especifica o número de vezes que será executada a instrução), *sql* (parâmetro que especifica a SQL que será executada). Parâmetros esses passados pela classe *Interface*;
- *selectSqlOleDb (Integer vezes, String sql),* responsável pela recuperação na base de dados usando o modo *OleDb,* por meio dos parâmetros: *vezes* (parâmetro que especifica o número de vezes que será executada a instrução), *sql* (parâmetro que especifica a SQL que será executada). Parâmetros esses passados pela classe *Interface*;

A classe ***Interface*** é responsável por todos os métodos relacionados à interação com o avaliador (quem realiza as execuções SQL). Ela implementa os métodos ***get*** e ***set*** de todos os campos declarados no *Frame* (janela onde os campos são mostrados para o testador) e também as ações dos botões, responsáveis pela ativação dos métodos. Na Figura 11 é apresentado um exemplo da interface gráfica da aplicação .NET.

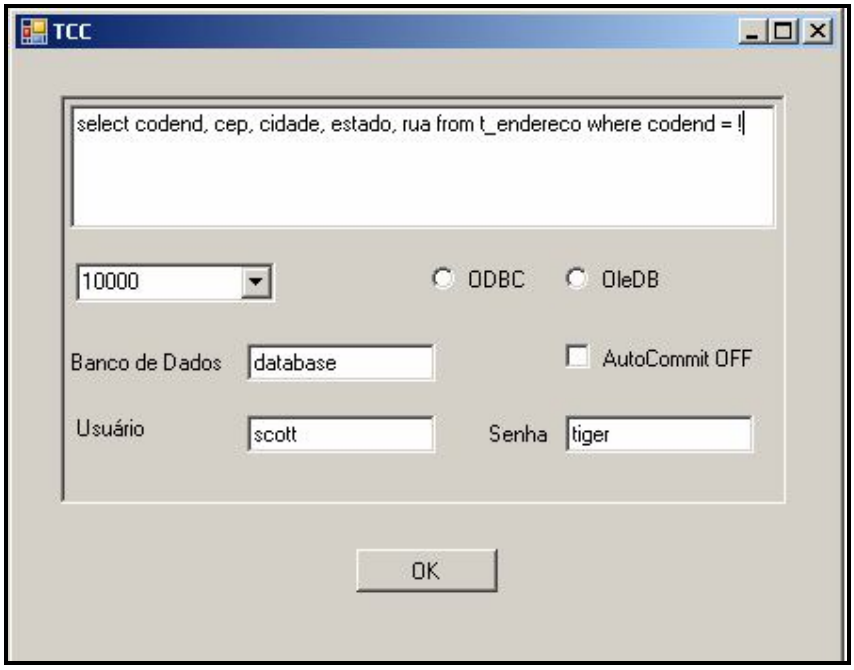


Figura 10 – Interface Gráfica da Aplicação .NET

A classe ***StopWatch*** não foi criada, diferente da aplicação Java, pois os métodos já estão implementados na própria linguagem.

2.3 Estratégias de Avaliações a serem exercitadas

Para a realização das avaliações, será criado um servidor com o SGBD Oracle9i instalado e configurado para estabelecer conexão com as aplicações. A estrutura do banco de dados será criada por meio de um *script*. O SGBD será otimizado para transações.

O objetivo desse trabalho será medir o desempenho entre as duas aplicações com o banco de dados OO Oracle9i.

2.3.1 Avaliação de inserção em uma tabela solitária

Esta avaliação constituirá em verificar o desempenho de inserção em ambas as aplicações em uma tabela, sem nenhuma referência a outra tabela. Para essa avaliação será utilizada a tabela **t_Endereço** do tipo **Endereço**, com os atributos: **codEnd**, **cep**, **cidade**, **estado** e **rua**.

A inserção nessa tabela será feita em número de vezes definidos nas aplicações. Tendo como números: 10, 100, 1000, 5000 e 10000.

As inserções serão efetuadas utilizando métodos diferentes de conexão com o SGBD Oracle9i, métodos esses: *Statement* e *PreparedStatement* para a aplicação Java, ambos JDBC, e para aplicação .NET os métodos ODBC e OleDb.

Essa inserção será realizada de duas maneiras: sem incremento e com incremento a chave primária, gerenciado pelas aplicações.

As aplicações terão um campo na interface gráfica, que permitirá que o *autocommit* de cada inserção esteja ativo ou não.

2.3.2 Avaliação de inserção em uma tabela com referência

Esta avaliação constituirá em verificar o desempenho de inserções em ambas as aplicações em uma tabela, com duas referências a outras tabelas. Para essa avaliação serão utilizadas as tabelas **t_clienteFisico** do tipo **clienteFisico**, com os atributos: **cpf** e **codCli**; a tabela **t_Endereço** do tipo **Endereço**, com os atributos: **codEnd, cep, cidade, estado** e **rua**; a tabela **t_Telefone** do tipo **Telefone**, com os atributos: **codtel** e **numeroTel**.

As inserções nessa tabela serão feitas em número de vezes definidos nas aplicações. Tendo como números: 10, 100, 1000, 5000 e 10000.

As inserções serão efetuadas utilizando métodos diferentes de conexão com o SGBD Oracle9i, métodos esses: *Statement* e *PreparedStatement* para a aplicação Java, ambos JDBC, e para aplicação .NET os métodos ODBC e OleDb.

Essa inserção será realizada de duas maneiras: sem incremento e com incremento a chave primária, gerenciado pelas aplicações.

As aplicações terão um campo na interface gráfica, que permitirá que o *autocommit* de cada inserção esteja ativo ou não.

2.3.3 Avaliação de atualização em uma tabela solitária

Esta avaliação constituirá em verificar o desempenho de atualização em ambas as aplicações em uma tabela, sem nenhuma referência a outra tabela. Para essa avaliação será utilizada a tabela **t_Endereço** do tipo **Endereço**, com os atributos: **codEnd**, **cep**, **cidade**, **estado** e **rua**.

As atualizações serão efetuadas utilizando métodos diferentes de conexão com o SGBD Oracle9i, métodos esses: *Statement* e *PreparedStatement* para a aplicação Java, ambos JDBC, e para aplicação .NET os métodos ODBC e OleDb.

Como na avaliação de inserção, será feito em número de vezes definidos pelas aplicações. Tendo como números de execuções: 10, 100, 1000, 5000 e 10000.

2.3.4 Avaliação de atualização em uma tabela com referência

Esta avaliação constituirá em verificar o desempenho de atualização em ambas as aplicações em uma tabela, com duas referências a outras tabelas. Para essa avaliação serão utilizadas as tabelas **t_clienteFisico** do tipo **clienteFisico**, com os atributos: **cpf** e **codCli**; a tabela **t_Endereço** do tipo **Endereço**, com os atributos: **codEnd, cep**, **cidade**, **estado** e **rua**; a tabela **t_Telefone** do tipo Telefone, com os atributos: **codtel** e **numeroTel**.

Também serão feito atualizações usando métodos diferentes de conexão com o SGBD Oracle9i, métodos esses: *Statement* e *PreparedStatement* para a aplicação Java, ambos JDBC, e para aplicação .NET os métodos ODBC e OleDb.

Como na avaliação de inserção, será feito em número de vezes definidos pelas aplicações. Tendo como números de execuções: 10, 100, 1000, 5000 e 10000.

2.3.5 Avaliação de recuperação em uma tabela solitária

Esta avaliação constituirá em verificar o desempenho de recuperações em ambas as aplicações em uma tabela sem nenhuma referência a outra tabela. Para essa avaliação será utilizada a tabela **t_Endereço** do tipo **Endereço**, com os atributos: **codEnd**, **cep**, **cidade**, **estado** e **rua**.

As recuperações serão efetuadas utilizando métodos diferentes de conexão com o SGBD Oracle9i, métodos esses: *Statement* e *PreparedStatement* para a aplicação Java, ambos JDBC, e para aplicação .NET os métodos ODBC e OleDb.

Como nas avaliações anteriores, será feito em número de vezes definidos pelas aplicações. Tendo como números de execuções: 10, 100, 1000, 5000 e 10000.

2.3.6 Avaliação de recuperação em uma tabela com referência

Esta avaliação constituirá em verificar o desempenho de recuperações em ambas as aplicações em uma tabela com duas referências a outras tabelas. Para essa avaliação serão utilizadas as tabelas **t_clienteFisico** do tipo **clienteFisico**, com os atributos: **cpf** e **codCli**; a tabela **t_Endereço** do tipo **Endereço**, com os atributos: **codEnd**, **cep, cidade**, **estado** e **rua**; a tabela **t_Telefone** do tipo Telefone, com os atributos: **codtel** e **numeroTel**.

As recuperações serão efetuadas utilizando métodos diferentes de conexão com o SGBD Oracle9i, métodos esses: *Statement* e *PreparedStatement* para a aplicação Java, ambos JDBC, e para aplicação .NET os métodos ODBC e OleDb.

Como nas avaliações anteriores, será feito em número de vezes definidos pelas aplicações. Tendo como números de execuções: 10, 100, 1000, 5000 e 10000.

2.3.7 Avaliação de exclusão em uma tabela solitária

Esta avaliação constituirá em verificar o desempenho de exclusões em ambas as aplicações em uma tabela sem nenhuma referência a outra tabela. Para essa avaliação será utilizada a tabela **t_Endereço** do tipo **Endereço**, com os atributos: **codEnd**, **cep**, **cidade**, **estado** e **rua**.

As exclusões serão efetuadas utilizando métodos diferentes de conexão com o SGBD Oracle9i, métodos esses: *Statement* e *PreparedStatement* para a aplicação Java, ambos JDBC, e para aplicação .NET os métodos ODBC e OleDb.

Como nas avaliações anteriores, será feito em número de vezes definidos pelas aplicações. Tendo como números de execuções: 10, 100, 1000, 5000 e 10000.

2.3.8 Avaliação de exclusão em uma tabela com referência

Esta avaliação constituirá em verificar o desempenho de exclusões em ambas as aplicações em uma tabela com duas referências a outras tabelas. Para essa avaliação serão utilizadas as tabelas **t_clienteFisico** do tipo **clienteFisico**, com os atributos: **cpf** e **codCli**; a tabela **t_Endereço** do tipo **Endereço**, com os atributos: **codEnd**, **cep**, **cidade**, **estado** e **rua**; a tabela **t_Telefone** do tipo Telefone, com os atributos: **codtel** e **numeroTel**.

As exclusões serão efetuadas utilizando métodos diferentes de conexão com o SGBD Oracle9i, métodos esses: *Statement* e *PreparedStatement* para a aplicação Java, ambos JDBC, e para aplicação .NET os métodos ODBC e OleDb.

Como nas avaliações anteriores, será feito em número de vezes definidos pelas aplicações. Tendo como números de execuções: 10, 100, 1000, 5000 e 10000.

2.3.9 Avaliação dos resultados a serem obtidos

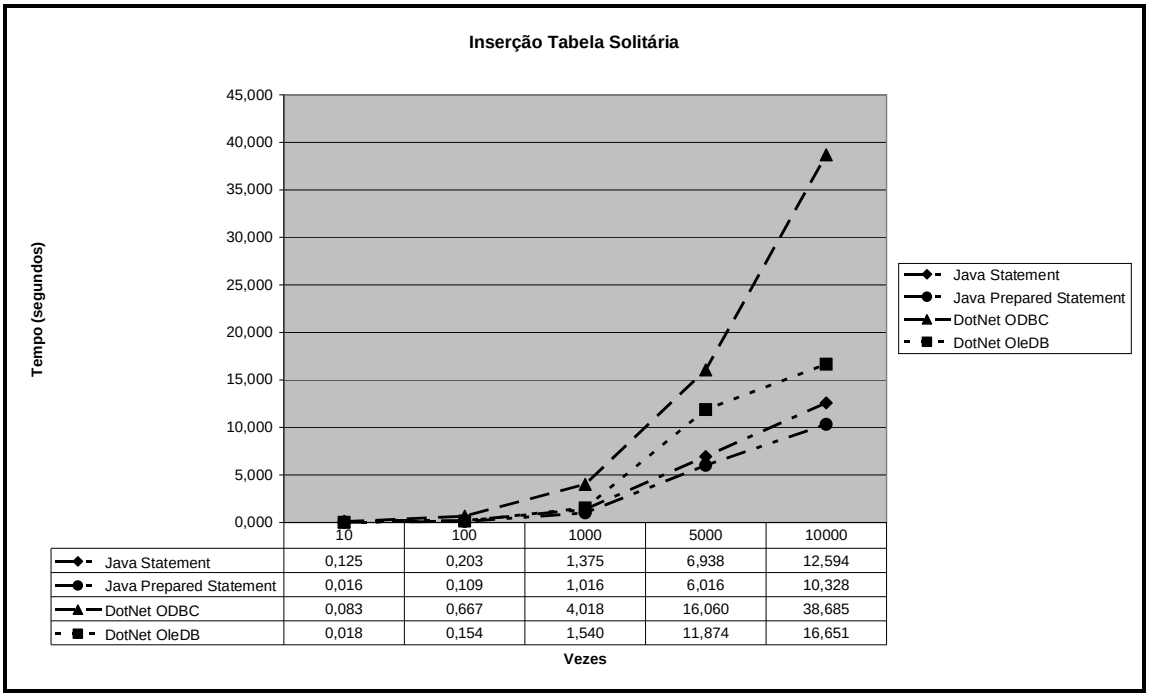
Os resultados obtidos nas avaliações serão comparados entre o número de vezes que foram executados e o tempo de execução para cada um dos casos, podendo assim, distinguir qual das aplicações tem o método de conexão com melhor desempenho em relação ao BDOO Oracle9i.

CAPÍTULO 3 – ESTUDO DE CASO

3.1 – Avaliação de inserção em uma tabela solitária

Foram realizadas seqüências de inserções de volumes de registros em uma tabela solitária, isto é, uma tabela que não tem referência com nenhuma outra tabela. O desempenho de cada aplicação foi avaliado de acordo com o tempo decorrido para que as inserções fossem concluídas. No Gráfico 1 são apresentados os resultados obtidos.

Gráfico 1 – Avaliação de Desempenho de Inserção em uma tabela solitária



	10	100	1000	5000	10000
Java Statement	0,125	0,203	1,375	6,938	12,594
Java Prepared Statement	0,016	0,109	1,016	6,016	10,328
DotNet ODBC	0,083	0,667	4,018	16,060	38,685
DotNet OleDB	0,018	0,154	1,540	11,874	16,651

Pode-se observar que a aplicação Java, utilizando o modo de execução *Prepared Statement*, tem desempenho superior a da aplicação .NET. Também se observou que incluindo um volume de dados menor, na casa de algumas centenas de registros, o modo de conexão OleDb da aplicação .NET obteve desempenho melhor que o modo de execução *Statement*, utilizado na aplicação Java, mas a medida que o número de registros aumenta, o desempenho do .NET diminui em uma proporção maior do que a do Java.

O modo de execução com pior desempenho foi o ODBC utilizado na aplicação .NET. Pode-se observar que quanto mais o número de registros aumenta, mais perceptível é o

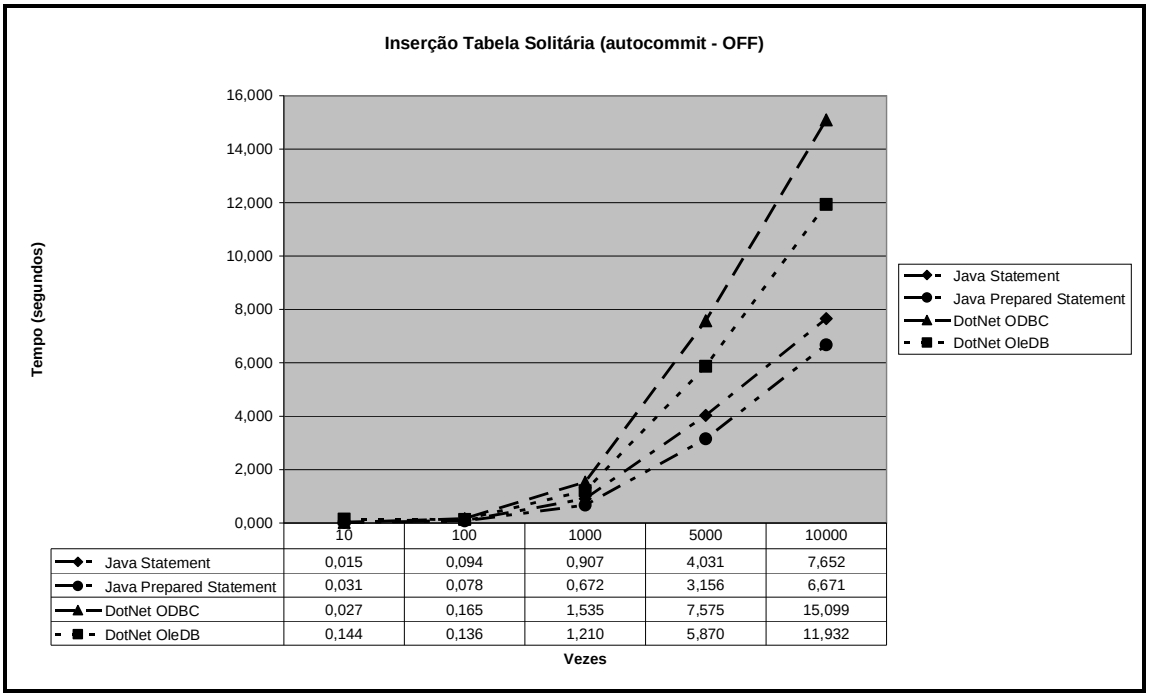
seu baixo desempenho, chegando a ser três vezes menor do que o modo de execução *Prepared Statement* utilizado na aplicação Java.

Nesta avaliação, a aplicação Java se mostra mais rápida que a aplicação .NET, tendo o modo de execução *Prepared Statement* sempre mais rápido que os outros métodos utilizados.

3.2 – Avaliação de inserção em uma tabela solitária com *autocommit* desativado

Para esta avaliação foi utilizado o método para desativar o *autocommit* das execuções. O desempenho de cada aplicação foi avaliado de acordo com o tempo decorrido para que as inserções fossem concluídas. No Gráfico 2 são apresentados os resultados obtidos.

Gráfico 2 – Avaliação de Desempenho de Inserção em uma tabela solitária com *autocommit* desativado



	10	100	1000	5000	10000
Java Statement	0,015	0,094	0,907	4,031	7,652
Java Prepared Statement	0,031	0,078	0,672	3,156	6,671
DotNet ODBC	0,027	0,165	1,535	7,575	15,099
DotNet OleDB	0,144	0,136	1,210	5,870	11,932

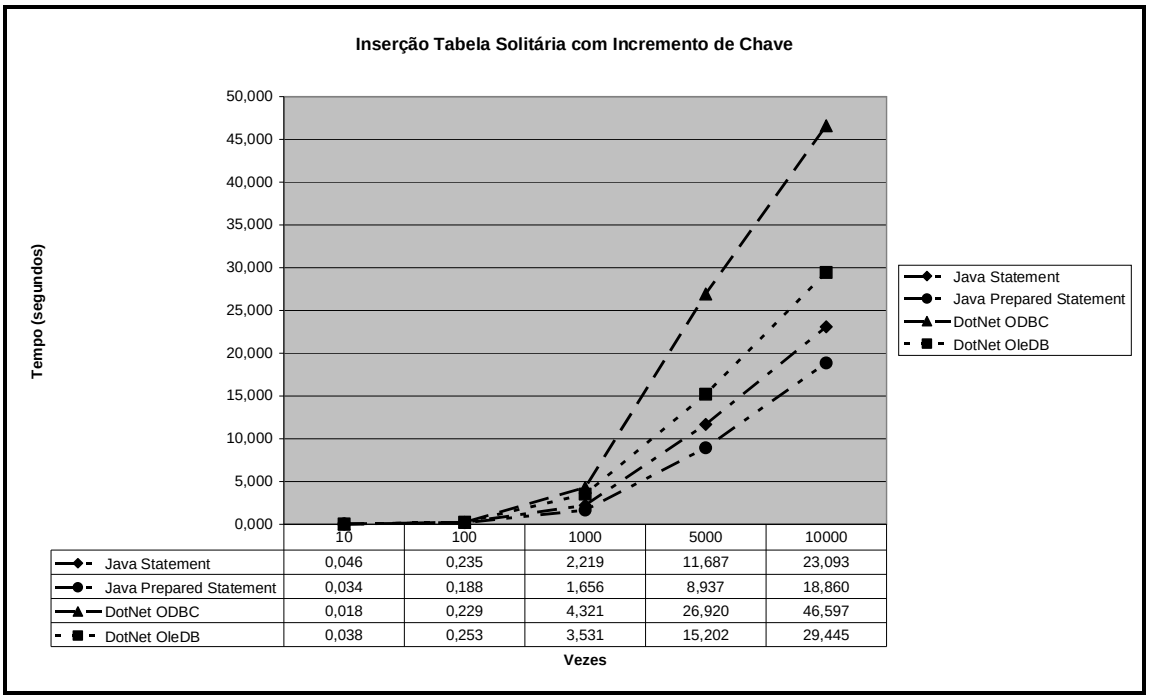
Como na avaliação anterior, o modo de execução *Prepared Statement* da aplicação Java tem melhor desempenho do que os modos de execução da aplicação .NET, e o modo mais lento é o modo de execução ODBC da aplicação .NET.

Nesta avaliação, a aplicação Java também se mostra mais rápida que a aplicação .NET, em geral, tendo o modo de execução *Prepared Statement* mais rápido que os outros métodos utilizados, sendo mais lento apenas em um volume de dados muito pequeno.

3.3 – Avaliação de inserção em uma tabela solitária com incremento de chave

Para esta avaliação foi utilizado a variável de controle do comando de repetição das *SQL*'s para que fosse feito o incremento da chave primária da tabela. O desempenho de cada aplicação foi avaliado de acordo com o tempo decorrido para que as inclusões fossem concluídas. No Gráfico 3 são apresentados os resultados obtidos.

Gráfico 3 – Avaliação de Desempenho de Inserção em uma tabela solitária com incremento de chave



	10	100	1000	5000	10000
Java Statement	0,046	0,235	2,219	11,687	23,093
Java Prepared Statement	0,034	0,188	1,656	8,937	18,860
DotNet ODBC	0,018	0,229	4,321	26,920	46,597
DotNet OleDB	0,038	0,253	3,531	15,202	29,445

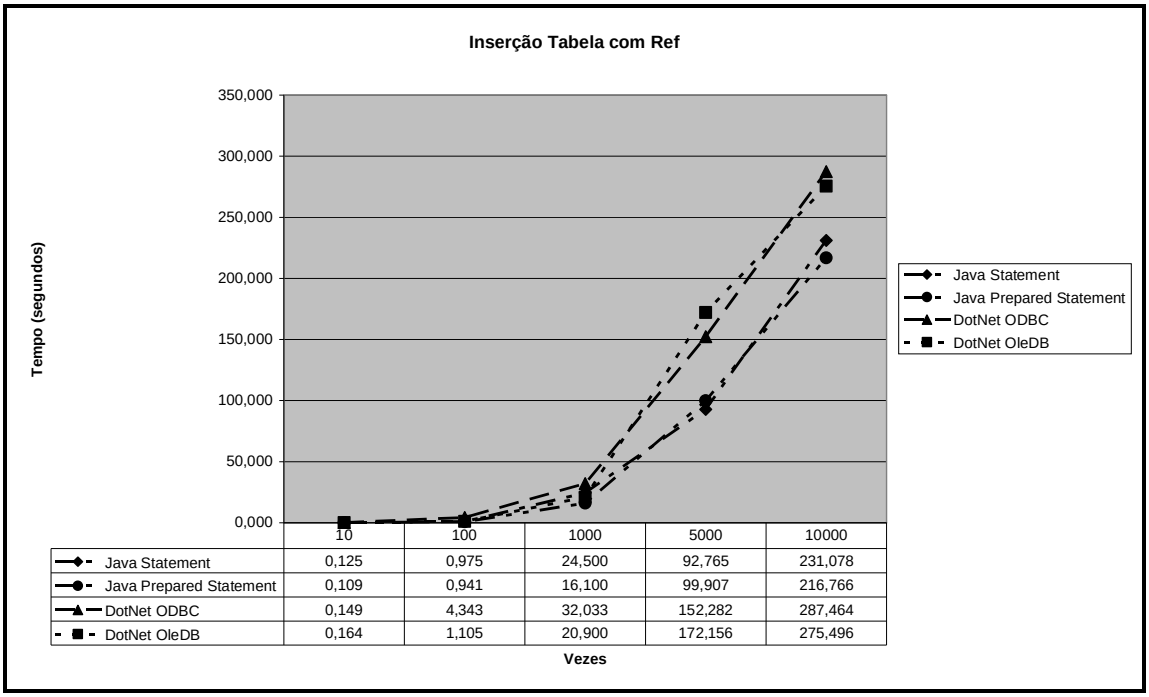
Nesta avaliação, o modo de execução da aplicação .NET tem melhor desempenho com algumas centenas de registros, mas ainda sim sendo o modo mais lento com um grande volume de registros.

O modo de execução *Prepared Statement* da aplicação Java, assim como nas avaliações anteriores, foi o método mais rápido com grande volume de registros.

3.4 – Avaliação de inserção em uma tabela com referência

Nesta avaliação foram realizadas inserções em uma tabela com duas referências a outras tabelas. Como nas avaliações anteriores, o desempenho foi avaliado de acordo com o tempo de inserção decorrido de cada aplicação. No Gráfico 4 são apresentados os resultados obtidos na inserção.

Gráfico 4 – Avaliação de Desempenho de Inserção em uma tabela com referência



	10	100	1000	5000	10000
Java Statement	0,125	0,975	24,500	92,765	231,078
Java Prepared Statement	0,109	0,941	16,100	99,907	216,766
DotNet ODBC	0,149	4,343	32,033	152,282	287,464
DotNet OleDB	0,164	1,105	20,900	172,156	275,496

Os resultados obtidos nesta avaliação demonstraram melhor desempenho dos métodos da aplicação Java.

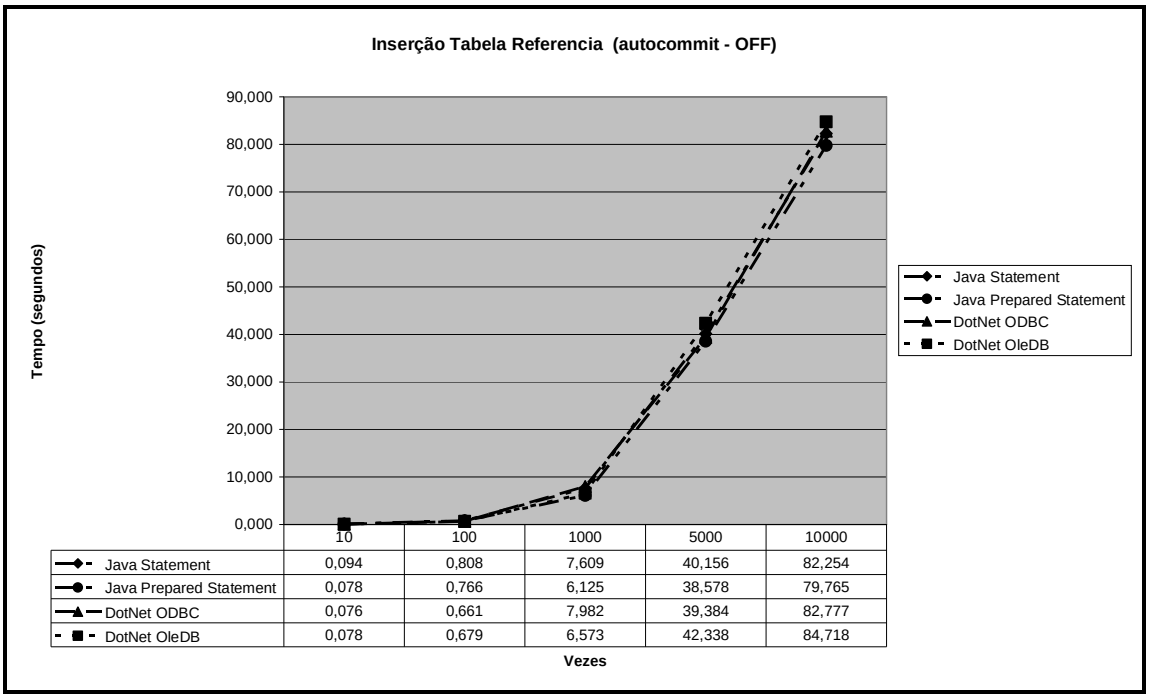
O método com melhor desempenho foi o *Prepared Statement* com exceção em algumas centenas e milhares de registros onde o método *Statement* obteve melhor desempenho, sendo ambos os métodos da aplicação Java.

O método com pior desempenho foi o ODBC, com exceção em alguns milhares de registros onde o método OleDb obteve desempenho mais baixo, sendo ambos utilizados no desenvolvimento da aplicação .NET.

3.5 – Avaliação de inserção em uma tabela com referência e *autocommit* desativado

Nesta avaliação foram realizadas inserções em uma tabela com duas referências a outras tabelas. Também foi utilizado o método para desativar o *autocommit* das execuções. O desempenho de cada aplicação foi avaliado de acordo com o tempo decorrido para que as inserções fossem concluídas. No Gráfico 5 são apresentados os resultados.

Gráfico 5 – Avaliação de Desempenho de Inserção em uma tabela com referência e *autocommit* desativado



	10	100	1000	5000	10000
Java Statement	0,094	0,808	7,609	40,156	82,254
Java Prepared Statement	0,078	0,766	6,125	38,578	79,765
DotNet ODBC	0,076	0,661	7,982	39,384	82,777
DotNet OleDB	0,078	0,679	6,573	42,338	84,718

Nesta avaliação os métodos de execução da aplicação .NET obtiveram desempenho sempre maior que os métodos da aplicação Java em algumas centenas de registros, mas em alguns milhares de registros o método *Prepared Statement* da aplicação Java obteve melhor desempenho.

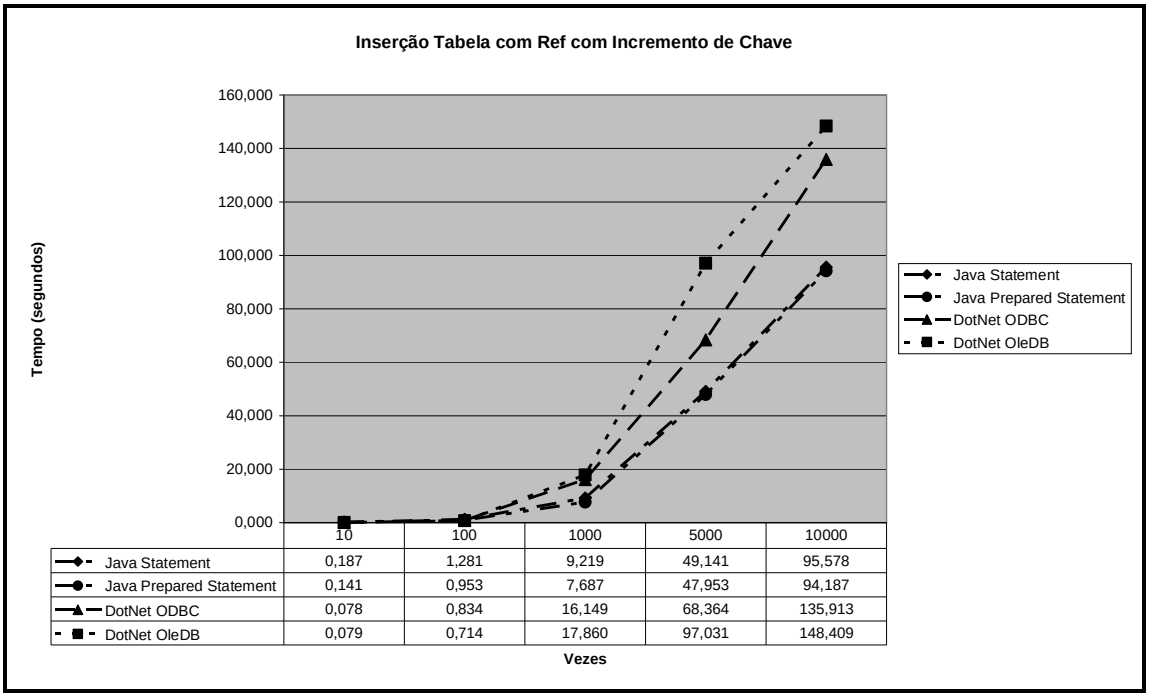
Entre os métodos demonstrados, o OleDb da aplicação .NET foi método com pior desempenho em alguns milhares de registros, tendo um bom desempenho com algumas centenas de registros.

3.6 – Avaliação de inserção em uma tabela com referência e incremento

Nesta avaliação foram realizadas inserções em uma tabela com duas referências a outras tabelas. Também foi utilizada a variável de controle do comando de repetição das

SQL's para que fosse feito o incremento da chave primária da tabela. O desempenho de cada aplicação foi avaliado de acordo com o tempo decorrido para que as inclusões fossem concluídas. No Gráfico 6 são apresentados os resultados obtidos.

Gráfico 6 – Avaliação de Desempenho de Inserção em uma tabela referência e incremento de chave



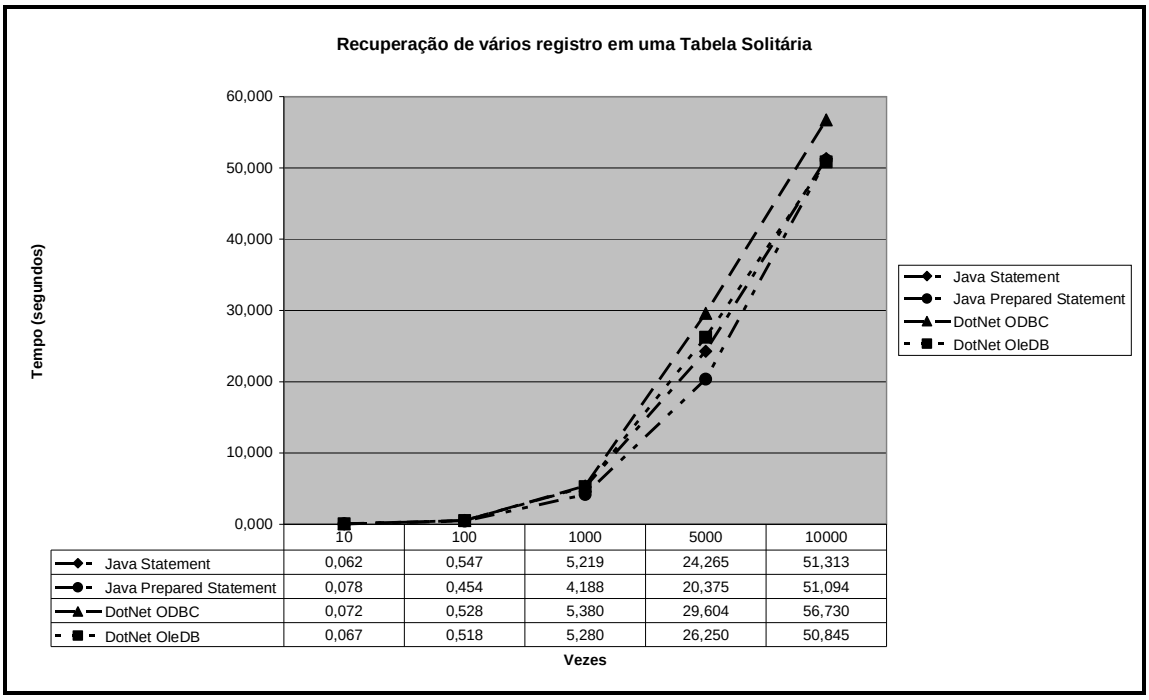
	10	100	1000	5000	10000
Java Statement	0,187	1,281	9,219	49,141	95,578
Java Prepared Statement	0,141	0,953	7,687	47,953	94,187
DotNet ODBC	0,078	0,834	16,149	68,364	135,913
DotNet OleDB	0,079	0,714	17,860	97,031	148,409

Nesta avaliação o modo de conexão ODBC da aplicação .NET obteve melhor desempenho com algumas centenas de registros, mas com alguns milhares de registros o modo de conexão *Prepared Statement* da aplicação Java foi o método com melhor desempenho tendo também o método *Statement* com um desempenho muito próximo. O método com o pior desempenho foi o OleDb da aplicação .NET.

3.7 – Avaliação de recuperação em uma tabela solitária

Nesta avaliação foram realizadas recuperações em uma tabela solitária. O desempenho de cada aplicação foi avaliado de acordo com o tempo decorrido para que as recuperações fossem concluídas. No Gráfico 7 são apresentados os resultados obtidos.

Gráfico 7 – Avaliação de Desempenho de Recuperação em uma tabela solitária



	10	100	1000	5000	10000
Java Statement	0,062	0,547	5,219	24,265	51,313
Java Prepared Statement	0,078	0,454	4,188	20,375	51,094
DotNet ODBC	0,072	0,528	5,380	29,604	56,730
DotNet OleDB	0,067	0,518	5,280	26,250	50,845

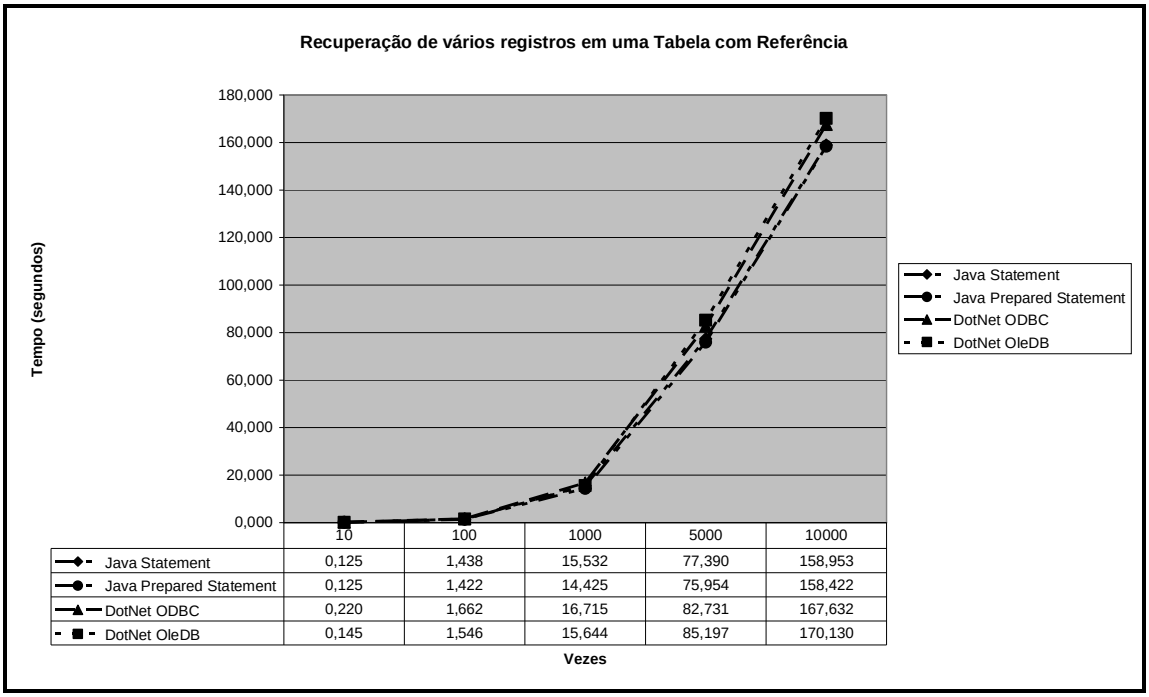
Nesta avaliação o método de execução *Prepared Statement* obteve melhor desempenho, com exceção apenas em um grande volume de dados onde os métodos OleDb e *Statement* obtiveram resultados equivalentes.

O método com pior desempenho foi o ODBC da aplicação .NET, com exceção em algumas centenas de registros.

3.8 – Avaliação de recuperação em uma tabela com referência

Nesta avaliação foram realizadas recuperações em uma tabela com referência a outras tabelas. O desempenho de cada aplicação foi avaliado de acordo com o tempo decorrido para que as recuperações fossem concluídas. No Gráfico 8 são apresentados os resultados obtidos.

Gráfico 8 – Avaliação de Desempenho de Recuperação em uma tabela com referência



	10	100	1000	5000	10000
Java Statement	0,125	1,438	15,532	77,390	158,953
Java Prepared Statement	0,125	1,422	14,425	75,954	158,422
DotNet ODBC	0,220	1,662	16,715	82,731	167,632
DotNet OleDB	0,145	1,546	15,644	85,197	170,130

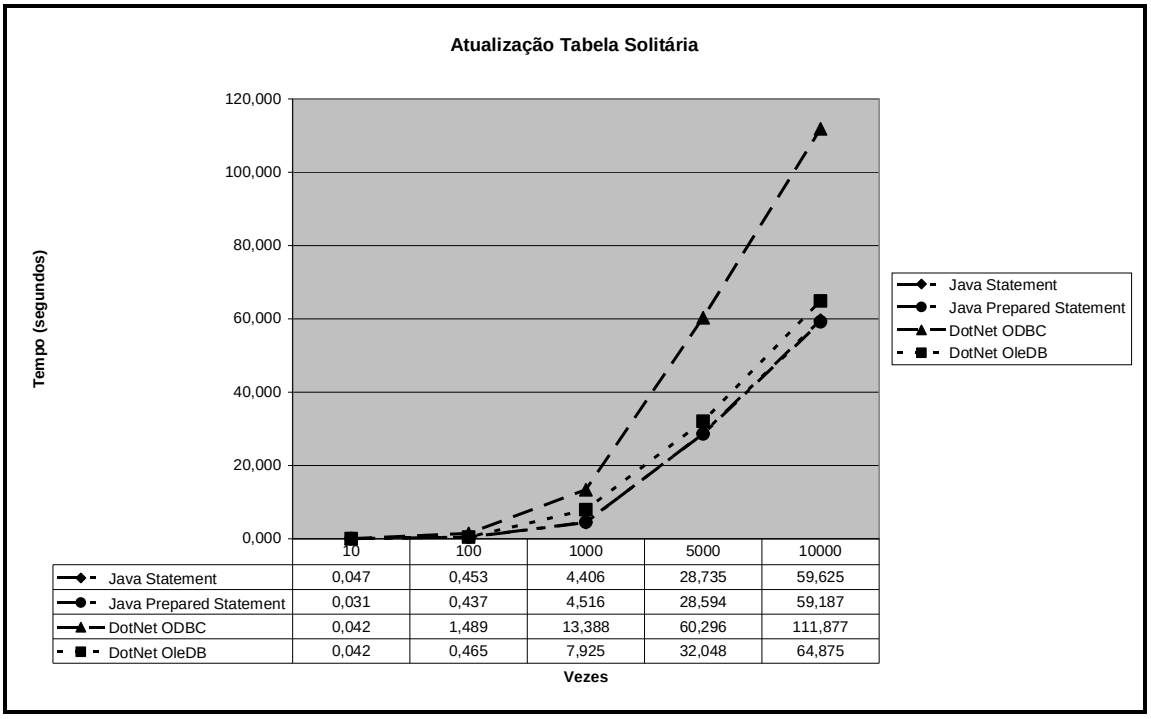
Nesta avaliação o método de conexão *Prepared Statement* obteve melhor desempenho, tendo também o método *Statement* com um desempenho muito próximo.

O método com pior desempenho foi o OleDb da aplicação .NET, com algumas exceções em que o método ODBC teve pior desempenho, mas ainda sim ambos os métodos obtiveram resultados satisfatórios.

3.9 – Avaliação de atualização em uma tabela solitária

Nesta avaliação foram realizadas atualizações em uma tabela solitária. O desempenho de cada aplicação foi avaliado de acordo com o tempo decorrido para que as atualizações fossem concluídas. No Gráfico 9 são apresentados os resultados obtidos nas atualizações.

Gráfico 9 – Avaliação de Desempenho de Atualização em uma tabela solitária



	10	100	1000	5000	10000
Java Statement	0,047	0,453	4,406	28,735	59,625
Java Prepared Statement	0,031	0,437	4,516	28,594	59,187
DotNet ODBC	0,042	1,489	13,388	60,296	111,877
DotNet OleDB	0,042	0,465	7,925	32,048	64,875

Esta avaliação apresentou dois métodos com melhor desempenho, o *Prepared Statement* e o *Statement*, ambos da aplicação Java.

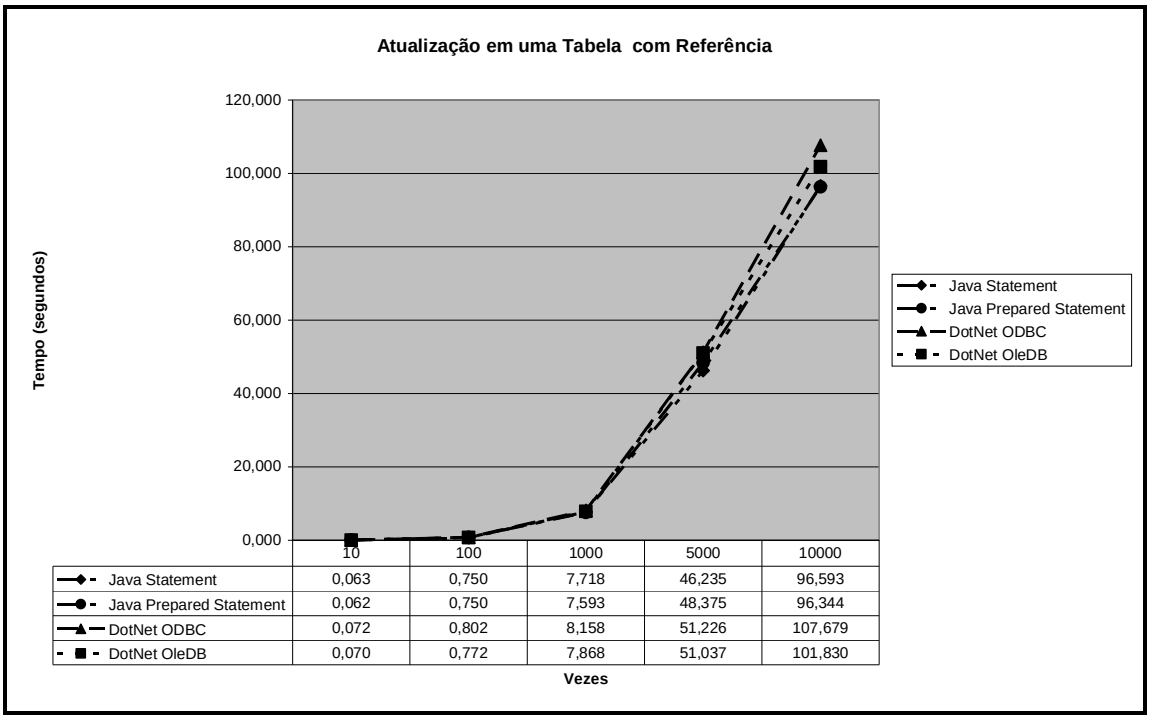
O método com pior desempenho, como nas avaliações anteriores, foi o método de conexão ODBC da aplicação .NET, que chegou a ter seu tempo de execução duas vezes maior que outros métodos.

O método OleDb da aplicação .NET obteve um resultado satisfatório, sempre tendo um desempenho próximo aos métodos com melhor desempenho.

3.10 – Avaliação de atualização em uma tabela com referência

Nesta avaliação foram realizadas atualizações em uma tabela com referência a outras tabelas. O desempenho de cada aplicação foi avaliado de acordo com o tempo decorrido para que as atualizações fossem concluídas. No Gráfico 10 são apresentados os resultados obtidos nas atualizações.

Gráfico 10 – Avaliação de Desempenho de Atualização em uma tabela com referência



	10	100	1000	5000	10000
Java Statement	0,063	0,750	7,718	46,235	96,593
Java Prepared Statement	0,062	0,750	7,593	48,375	96,344
DotNet ODBC	0,072	0,802	8,158	51,226	107,679
DotNet OleDB	0,070	0,772	7,868	51,037	101,830

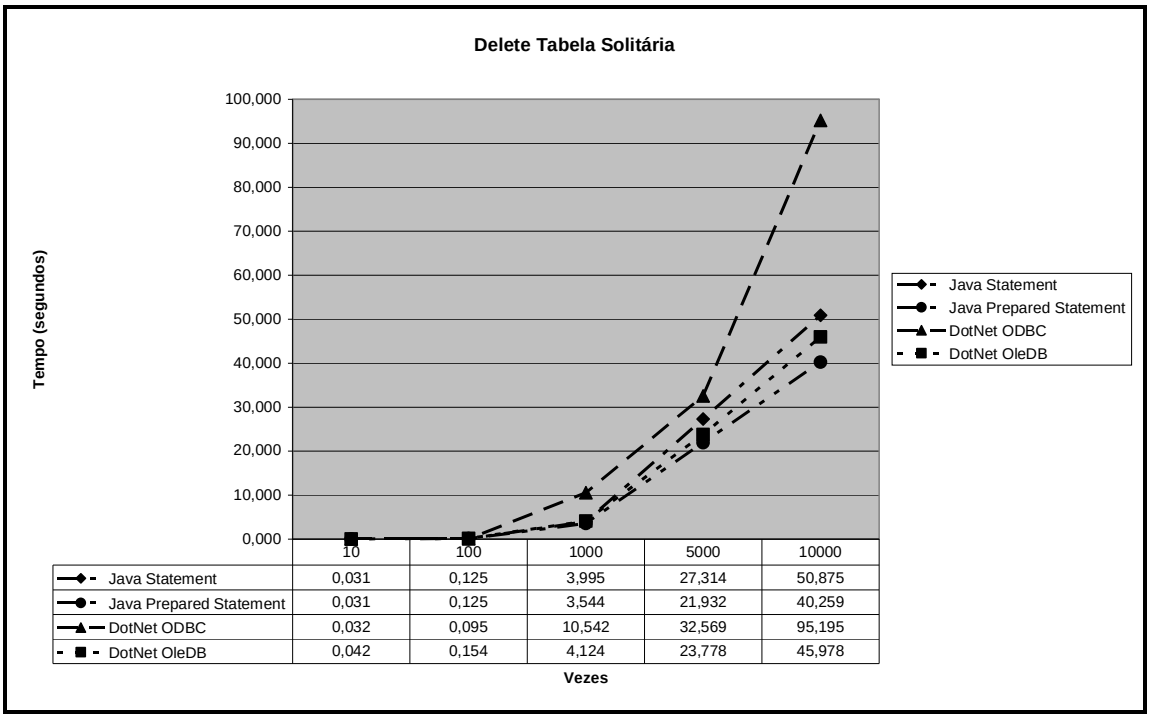
Nesta avaliação os métodos das duas aplicações obtiveram resultados parecidos, tendo o método *Prepared Statement* com melhor desempenho, sendo alguns segundos mais rápidos que os outros métodos.

O método com pior desempenho, como nas avaliações anteriores, foi o método ODBC da aplicação .NET. Ele foi alguns segundos mais lento que os outros métodos, mas ainda sim com um desempenho satisfatório.

3.11 – Avaliação de exclusão em uma tabela solitária

Nesta avaliação foram realizadas exclusões em uma tabela solitária. O desempenho de cada aplicação foi avaliado de acordo com o tempo decorrido para que as exclusões fossem concluídas. No Gráfico 11 são apresentados os resultados obtidos.

Gráfico 11 – Avaliação de Desempenho de Exclusão em uma tabela solitária



	10	100	1000	5000	10000
Java Statement	0,031	0,125	3,995	27,314	50,875
Java Prepared Statement	0,031	0,125	3,544	21,932	40,259
DotNet ODBC	0,032	0,095	10,542	32,569	95,195
DotNet OleDB	0,042	0,154	4,124	23,778	45,978

Nesta avaliação o método com melhor desempenho foi o *Prepared Statement* da aplicação Java, mantendo seu desempenho sempre constante, com poucos ou vários registros.

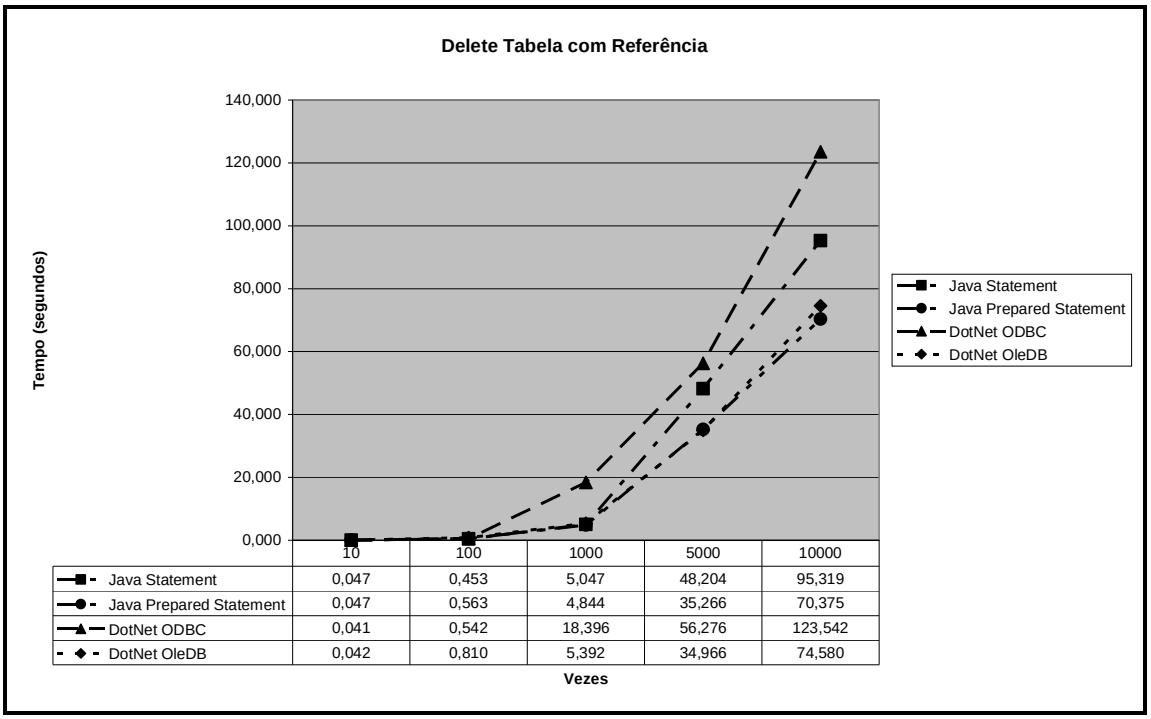
O método com pior desempenho foi o ODBC da aplicação .NET, sendo até duas vezes mais lento que o método *Prepared Statement.*

Com um desempenho satisfatório mostrou-se o método OleDb da aplicação .NET, mantendo seu desempenho constante.

3.12 – Avaliação de exclusão em uma tabela com referência

Nesta avalação foram realizadas exclusões em uma tabela com referência a outras tabelas. O desempenho de cada aplicação foi avaliado de acordo com o tempo decorrido para que as exclusões fossem concluídas. No Gráfico 12 são apresentados os resultados obtidos.

Gráfico 12 – Avaliação de Desempenho de Exclusão em uma tabela com referência



	10	100	1000	5000	10000
Java Statement	0,047	0,453	5,047	48,204	95,319
Java Prepared Statement	0,047	0,563	4,844	35,266	70,375
DotNet ODBC	0,041	0,542	18,396	56,276	123,542
DotNet OleDB	0,042	0,810	5,392	34,966	74,580

Nesta avaliação, como na anterior, o método *Prepared Statement* da aplicação Java obteve melhor desempenho dentre todos os métodos, tendo também o método OleDb da aplicação .NET com um desempenho muito próximo.

O método com pior desempenho foi o ODBC da aplicação .NET, obtendo um desempenho não satisfatório a partir de alguns milhares de registros.

3.13 – Análise Geral

A Tabela 1 foi construída de acordo com os resultados apresentados nesse capítulo. A primeira coluna específica a avaliação, e as demais colunas apresentam a comparação entre os resultados obtidos.

Tabela 1 – Resumo Comparativo dos Métodos de Conexão com o SGBD Oracle9i

Descrição	*Statement*	*Prepared Statement*	ODBC	OleDb
Avaliação de Desempenho de Inserção em uma tabela Solitária (100 registros)	bom	ótimo	ruim	regular
Avaliação de Desempenho de Inserção em uma tabela Solitária (10000 registros)	regular	ótimo	ruim	bom
Avaliação de Desempenho de Inserção em uma tabela Solitária com *autocommit* desativado (100 registros)	bom	ótimo	ruim	regular
Avaliação de Desempenho de Inserção em uma tabela Solitária com *autocommit* desativado (10000 registros)	bom	ótimo	ruim	regular
Avaliação de Desempenho de Inserção em uma tabela Solitária com incremento de chave (100 registros)	regular	ótimo	ruim	bom
Avaliação de Desempenho de Inserção em uma tabela Solitária com incremento de chave (10000 registros)	bom	ótimo	ruim	regular
Avaliação de Desempenho de Inserção em uma tabela com duas Referências a outras tabelas (100 registros)	bom	ótimo	ruim	regular
Avaliação de Desempenho de Inserção em uma tabela com duas Referências a outras tabelas (10000 registros)	bom	ótimo	ruim	regular
Avaliação de Desempenho de Inserção em uma tabela com Referência e *autocommit* desativado (100 registros)	ruim	regular	melhor	bom
Avaliação de Desempenho de Inserção em uma tabela com Referência e *autocommit* desativado (10000 registros)	bom	ótimo	regular	ruim
Avaliação de Desempenho de Inserção em uma tabela Referência e incremento de chave (100 registros)	ruim	regular	bom	melhor
Avaliação de Desempenho de Inserção em uma tabela Referência e incremento de chave (10000 registros)	bom	ótimo	regular	ruim

Tabela 2 – Resumo Comparativo dos Métodos de Conexão com o SGBD Oracle9i (continuação)

Descrição	*Statement*	*Prepared Statement*	ODBC	OleDb
Avaliação de recuperação em uma tabela solitária (100 registros)	ruim	ótimo	bom	regular
Avaliação de recuperação em uma tabela solitária (10000 registros)	regular	bom	ruim	ótimo
Avaliação de Desempenho de Recuperação em uma tabela com Referência (100 registros)	bom	ótimo	ruim	regular
Avaliação de Desempenho de Recuperação em uma tabela com Referência (10000 registros)	bom	ótimo	regular	ruim
Avaliação de Desempenho de Atualização em uma tabela Solitária (100 registros)	bom	ótimo	ruim	regular
Avaliação de Desempenho de Atualização em uma tabela Solitária (10000 registros)	bom	ótimo	ruim	regular
Avaliação de Desempenho de Atualização em uma tabela com Referência (100 registros)	bom	ótimo	ruim	regular
Avaliação de Desempenho de Atualização em uma tabela com Referência (10000 registros)	bom	ótimo	ruim	regular
Avaliação de Desempenho de Exclusão em uma tabela Solitária (100 registros)	bom	ótimo	regular	pior
Avaliação de Desempenho de Exclusão em uma tabela Solitária (10000 registros)	bom	ótimo	ruim	regular
Avaliação de Desempenho de Exclusão em uma tabela com Referência (100 registros)	ótimo	regular	bom	ruim
Avaliação de Desempenho de Exclusão em uma tabela com Referência (10000 registros)	regular	ótimo	ruim	bom

Conforme mostrado nas Tabelas 1 e 2, pode-se concluir que a escolha do modo de conexão utilizado com o SGBD Oracle9i apresenta desempenho diferente dependendo de dois fatores: da quantidade de dados e da quantidade de referências que são processados.

CONCLUSÃO E TRABALHOS FUTUROS

O objetivo desse trabalho foi apresentar um estudo comparativo entre duas aplicações desenvolvidas em plataformas OO distintas, Java e .NET, utilizando-se o SGBD Oraclei9, para a realização das avaliações de inserção, recuperação, atualização e exclusão.

A partir dos resultados apresentados no capítulo anterior, são esboçadas as seguintes conclusões.

i) O método de execução *Prepared Statement* utilizado na aplicação Java esteve sempre entre os métodos com melhor desempenho em todas as avaliações de inserção, recuperação e atualização, sendo uma boa escolha no desenvolvimento de aplicações na linguagem Java;

ii) O método de execução *Statement*, também utilizado na aplicação Java, obteve um desempenho sempre próximo ao método *Prepared Statement*. Em algumas exceções este método obteve um melhor desempenho, mas isso sempre em volume de dados menores, sendo uma boa escolha para aplicações que utilizem um baixo volume de dados;

iii) O método de conexão OleDb da aplicação .NET demonstrou um desempenho satisfatório. Ele sempre obteve desempenho próximo ao método *Statement*, mas ainda sim com um desempenho menor em relação aos métodos da aplicação Java;

iv) Dentre os métodos utilizados nas aplicações, pode-se concluir que o método de conexão ODBC da aplicação .NET foi o método que obteve pior desempenho dentre todos os métodos, chegando a ser três vezes mais lento que os outros métodos na avaliação de inserção em tabelas solitárias.

v) Conclui-se, também, que o método de conexão utilizado no momento do desenvolvimento de uma aplicação, pode fazer diferença na comunicação com o SGBD Oracle9i. Como se observou, o método *Prepared* Statement, utilizado na aplicação Java, obteve melhor desempenho dentre os métodos utilizados no desenvolvimento das aplicações utilizadas nas avaliações. Na aplicação .NET o método OleDb também se mostrou a melhor escolha para a comunicação com o SGBD Oracle9i, tendo um desempenho melhor em relação ao método ODBC.

Para trabalhos futuros, pode-se utilizar outros métodos de conexão com o SGBD Oracle9i, como exemplo o *framework Hibernate*, que é utilizado para fazer o mapeamento objeto relacional em aplicações Java, sendo também disponibilizado para o .NET com o nome de *NHibernate*. Também há a opção de realizar as mesmas avaliações em outros SGBD's com suporte a OO.

REFERÊNCIAS

Cornell, G. e Horstmann, C. S. Core Java 2. São Paulo: Makron Books. 2004. 823 p.

Elmasri, R. e S. Navathe. Sistemas de Banco de Dados. São Paulo: Addison-Wesley. 2005. 724 p.

Jia, X. Object-oriented software in Java : principles, patterns, and frameworks. Nova York: Addison-Wesley Longman. 2000. 507 p.

Korth, H. F., Silberschatz, A. et al. Sistema de Banco de Dados. São Paulo: Makron Books. 1999. 778 p.

Pressman, R. S. Engenharia de Software. São Paulo: Makron Books. 1995. 1056 p.

Patton, R. Software Testing, SAMS. 2001.

Windom, J., Garcia-Molina, H. et al. Implementação de Sistemas de Bancos de Dados. Rio de Janeiro: Campus. 2001. 687 p.

Filho, S., Vieira, O., Zara, P. M. Microsoft .net : uma visão geral para programadores. São Paulo: SENAC, 2002.

Lam, H., Thai, T. L. NET Framework Essentials, 3rd Edition. O'Reilly. Agosto 2003. 384 p.

Sousa, Mauro Sérgio Ribeiro. Mineração de Dados: Uma implementação fortemente acoplada a um sistema gerenciador de banco de dados paralelo [Rio de Janeiro] 1998 VIII, 67 p., 29, 7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1998)

Muller, Alex Moreira. PAPEL DE UMA ARQUITETURA MODULAR NO DESENVOLVIMENTO DISTRIBUÍDO DE SOFTWARE, UM ESTUDO DE CASO NO YART. 27 f. Monografia (Pós Graduação em Ciência da Computação) - UFR, Minas Gerais, 2006.

Delfino, Sérgio Roberto. Avaliação de Sistemas Gerenciadores de Banco de Dados Gratuitos para Aplicação em Sistemas de Gerenciamento de Imagens Médicas. Monografia (Graduação em Ciência da Computação) - UNIVEM, Marília, 2004.

Sharp, J., Longshaw, A., Roxburgh, P. **Microsoft Visual J# .NET.** Microsoft Press. Setembro 2002. 944 p.

Booch, G; Jacobson, I; Rumbaugh, J. **. UML: guia do usuário**. 2ª ed. Rio de Janeiro: Elsevier, 2006. 472p.

Lam, H., Thai, T. L. Net Framework Essentials, 3rd Edition. O`Reilly. Agosto 2003. 384 p.

Ruiz, F. H. M. e Gonçalves, G. de S. Desenvolvimento de Sistema de Banco de Dados Orientado a Objetos com Apoio da UML. 107 f. Monografia (Graduação em Ciência da Computação) – UNIVEM, Marília, 2007.

Sierra, K. e Bates, B. **Use a Cabeça! Java**, 2ª Edição. O'Reilly. 2007.

Object-oriented database management system (OODBMS) definition. Disponível em: <http://www.service-architecture.com/object-oriented-databases/articles/object-oriented_database_oodbms_definition.html>. Data de acesso: 14/03/2008.

The Object-Oriented Database Manifesto. Disponível em <http://www.cs.cmu.edu/afs/cs.cmu.edu/user/clamen/OODBMS/Manifesto/htManifesto/Manifesto.html>. Data de acesso: 15/03/2008.

MSDN 1. Disponível em <http://msdn.microsoft.com/en-us/library/w0x726c2.aspx> Data de acesso: 21/06/2008

MSDN 2. Disponível em <http://msdn.microsoft.com/en-us/library/ms754130.aspx>. Data de acesso: 21/06/2008.

MSDN 3. Disponível em <http://msdn.microsoft.com/en-us/netframework/aa663324.aspx>. Data de acesso: 21/06/2008.

MSDN 4. Disponível em <http://msdn.microsoft.com/en-us/netframework/aa663328.aspx> Data de acesso: 21/06/2008.

MSDN 5. Disponível em <http://msdn.microsoft.com/en-us/netframework/aa663320.aspx> Data de acesso: 21/06/2008.

Introducing the .NET Framework 3.5. Disponível em <http://download.microsoft.com/download/f/3/2/f32ff4c6-174f-4a2f-a58f-ed28437d7b1e/Introducing_NET_Framework_35_v1.doc>. Data de acesso: 08/10/2008.

APÊNDICE A – APLICAÇÃO JAVA

Neste apêndice é apresentado o código das classes implementadas na aplicação JAVA, utilizadas na execução das avaliações realizadas neste trabalho, sendo apresentados apenas os métodos sem as descrições, pois foram descritos anteriormente.

Classe ConexãoDB

```
package tcc;

import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.*;
import java.io.*;
import oracle.jdbc.driver.*;
import oracle.sql.*;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import javax.swing.JOptionPane;

public class ConexaoDB {

   private Statement stmt;
   private Connection con; // objeto de conexão
   StopWatch cron = new StopWatch(); // objeto instanciado da classe StopWatch()

   //método usado para registrar o driver oracle
   public void Conectar() {
      try {

         DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

      } catch (SQLException x) {
```

```
                JOptionPane.showMessageDialog(null, x.getMessage());
            }
        }


        //método usado para conectar o BD através
        //da entrada de parametros obtidos de outra classe
        public void getConexao(String bd, String user, String pass) {
            try {
                Conectar();
                con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:" +
bd, user, pass);
                System.out.println("Conectado ao BD");
            } catch (SQLException x) {
                JOptionPane.showMessageDialog(null, "Não Foi Concluída a Conexão Com
Base de Dados");
                return;
            }
        }


        //método usado para fechar a conexão com o BD
        public void CloseConexao() {
            try {
                con.close();
            } catch (SQLException x) {
                JOptionPane.showMessageDialog(null, "Base de Dados Está Aberta na
Memoria");
            }
        }


        //Utilização de PreparedStatement
```

```
        //método usado para a execução SQL(INSERT), parametros obtidos de outra classe
        public void insertSql(Boolean autocomm, Integer vezes, String sql) {
            PreparedStatement save = null;
            try {

                cron.start(); // pega o tempo corrente
                save = con.prepareStatement(sql);
                if (autocomm == true) {
                    con.setAutoCommit(false);
                }
                for (int i = 1; i <= vezes; i++) {

                    save.executeUpdate();
                }
                if (autocomm == true) {
                    con.commit();
                    con.setAutoCommit(true);
                }
                save.close();
                CloseConexao();
                cron.stop(); // pega o tempo corrente
                JOptionPane.showMessageDialog(null, "Tempo de inserção foi de: " + cron.getElapsedTimeSecs() + " segundos");
            } catch (SQLException ex) {
                Logger.getLogger(ConexaoDB.class.getName()).log(Level.SEVERE, null, ex);
            }

        }


        //Utilização de Statement
```

```
        //método usado para a execução SQL(INSERT), parametros obtidos de outra
classe
        public void insertSqlStat(Boolean autocomm, Integer vezes, String sql) {
            Statement save = null;
            try {

                cron.start(); // pega o tempo corrente
                save = con.createStatement();
                if (autocomm == true) {
                    con.setAutoCommit(false);
                }
                for (int i = 1; i <= vezes; i++) {
                    save.executeUpdate(sql);
                }
                if (autocomm == true) {
                    con.commit();
                    con.setAutoCommit(true);
                }
                save.close();
                CloseConexao();
                cron.stop(); // pega o tempo corrente
                JOptionPane.showMessageDialog(null, "Tempo de inserção foi de: " +
cron.getElapsedTimeSecs() + " segundos");
            } catch (SQLException ex) {
                Logger.getLogger(ConexaoDB.class.getName()).log(Level.SEVERE,    null,
ex);
            }


        }

        public void execincrSql(Boolean autocomm, Integer vezes, String sql) {
            sql = sql.replace('?', '!');
            PreparedStatement save = null;
```

```
            try {

                cron.start(); // pega o tempo corrente

                if (autocomm == true) {
                    con.setAutoCommit(false);
                }
                for (int i = 1; i < vezes; i++) {
                    save = con.prepareStatement(sql);
                    //save.setInt(1, i);
                    //save.executeUpdate();
                    save.executeUpdate(sql.replace("!", String.valueOf(i)));
                    save.close();
                }
                if (autocomm == true) {
                    con.commit();
                    con.setAutoCommit(true);
                }

                CloseConexao();
                cron.stop(); // pega o tempo corrente
                JOptionPane.showMessageDialog(null, "Tempo de execução foi de: " +
cron.getElapsedTimeSecs() + " segundos");
            } catch (SQLException ex) {
                Logger.getLogger(ConexaoDB.class.getName()).log(Level.SEVERE,  null,
ex);
            }

        }

        //Utilização de Statement
        //método usado para a execução SQL(INSERT), parametros obtidos de outra
classe
        public void execincrSqlStat(Boolean autocomm, Integer vezes, String sql) {
```

```
            sql = sql.replace('?', '!');
            Statement save = null;
            try {

                cron.start(); // pega o tempo corrente

                if (autocomm == true) {
                    con.setAutoCommit(false);
                }
                for (int i = 1; i < vezes; i++) {
                    save = con.createStatement();
                    save.executeUpdate(sql.replace("!", String.valueOf(i)));
                    save.close();
                }
                if (autocomm == true) {
                    con.commit();
                    con.setAutoCommit(true);
                }
                CloseConexao();
                cron.stop(); // pega o tempo corrente
                JOptionPane.showMessageDialog(null, "Tempo de execução foi de: " +
cron.getElapsedTimeSecs() + " segundos");
            } catch (SQLException ex) {
                Logger.getLogger(ConexaoDB.class.getName()).log(Level.SEVERE,    null,
ex);
            }


        }


        public void selectSqlSta(Integer vezes, String sql) {
            sql = sql.replace('?', '!');
            Statement save;
```

```
            try {
                cron.start();
                for (int i = 1; i < vezes; i++) {
                    save = con.createStatement();
                    ResultSet rs = save.executeQuery(sql.replace("!", String.valueOf(i)));
                    while (rs.next()) {

                    }
                    save.close();
                }
                CloseConexao();
                cron.stop();
                JOptionPane.showMessageDialog(null, "Tempo de seleção Statement foi de:
" + cron.getElapsedTimeSecs() + " segundos");
            } catch (SQLException ex) {
                Logger.getLogger(ConexaoDB.class.getName()).log(Level.SEVERE,    null,
ex);
            }

        }

        public void selectSql(Integer vezes, String sql) {
            sql = sql.replace('?', '!');
            PreparedStatement save;
            try {
                cron.start();
                for (int i = 1; i < vezes; i++) {
                    save = con.prepareStatement((sql.replace("!", String.valueOf(i))));
                    ResultSet rs = save.executeQuery();
                    while (rs.next()) {

                    }
                    save.close();
```

```
                }
                CloseConexao();
                cron.stop();
                JOptionPane.showMessageDialog(null, "Tempo de seleção
PreparedStatement foi de: " + cron.getElapsedTimeSecs() + " segundos");
            } catch (SQLException ex) {
                Logger.getLogger(ConexaoDB.class.getName()).log(Level.SEVERE, null,
ex);
            }

        }

    }
```

Classe Interface

```
/*
 * Interface.java
 *
 * Created on 11 de Junho de 2008, 12:28
 */
package tcc;

import javax.swing.JOptionPane;

/**
 *
 * @author  Diego
 */
public class Interface extends javax.swing.JFrame {

   ConexaoDB conDB = new ConexaoDB(); // objeto instanciado da classe ConexaoDB()
    private String escolha = "0";
    private String sql;

    /** Creates new form Interface */
    public Interface() {
        initComponents();
        setChoose(); // inicializa Choice()
    }

    //método para inicializar Choice()
    public void setChoose() {
        choose.add("1");
        choose.add("10");
        choose.add("100");
        choose.add("1000");
```

```
    choose.add("5000");
    choose.add("10000");
}

//método usado para definir o numero de vezes da instrução SQL
public String getVezes() {
    if (choose.getSelectedItem().equals(" ")) {
        JOptionPane.showMessageDialog(this, "Escolha o numero de vezes");
    } else {
        this.escolha = choose.getSelectedItem();
    }
    return this.escolha;
}

//método usado para obter a instrução SQL
public String getSql() {
    this.sql = TextSQL.getText();
    return this.sql;
}

//método de evento do botão Deletar
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    jPanel1 = new javax.swing.JPanel();
    jScrollPane1 = new javax.swing.JScrollPane();
    TextSQL = new javax.swing.JTextArea();
    choose = new java.awt.Choice();
    bdfield = new javax.swing.JTextField();
    userfield = new javax.swing.JTextField();
    passfield = new javax.swing.JTextField();
    statement = new javax.swing.JRadioButton();
    prestatement = new javax.swing.JRadioButton();
    commitSel = new javax.swing.JCheckBox();
```

```
        Gravar = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("TCC");

        jPanel1.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(0, 0, 0), 2,
true));

        TextSQL.setColumns(20);
        TextSQL.setRows(5);
        jScrollPane1.setViewportView(TextSQL);

        bdfield.setBorder(javax.swing.BorderFactory.createTitledBorder("Banco de Dados"));

        userfield.setBorder(javax.swing.BorderFactory.createTitledBorder("Usuário"));

        passfield.setBorder(javax.swing.BorderFactory.createTitledBorder("Senha"));

        statement.setText("Statement");
        statement.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                statementActionPerformed(evt);
            }
        });

        prestatement.setText("PrepareStatement");
        prestatement.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                prestatementActionPerformed(evt);
            }
        });

        commitSel.setText("AutoCommit OFF");
```

```
        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                    .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 487,
Short.MAX_VALUE)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(userfield, javax.swing.GroupLayout.PREFERRED_SIZE,
153, javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                        .addComponent(passfield, javax.swing.GroupLayout.PREFERRED_SIZE,
153, javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGroup(jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                        .addComponent(choose, javax.swing.GroupLayout.PREFERRED_SIZE,
98, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(bdfield, javax.swing.GroupLayout.PREFERRED_SIZE,
153, javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGap(85, 85, 85)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                        .addComponent(commitSel)
                        .addGroup(jPanel1Layout.createSequentialGroup()
                            .addComponent(statement)
                            .addGap(18, 18, 18)
```

```
                    .addComponent(prestatement)))))
            .addContainerGap())
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(25, 25, 25)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(choose, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(36, 36, 36)
                        .addComponent(bdfield, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGroup(jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASEL
INE)
                            .addComponent(statement)
                            .addComponent(prestatement))
                        .addGap(18, 18, 18)
                        .addComponent(commitSel)))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASEL
INE)
```

```
                .addComponent(userfield, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(passfield, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addContainerGap(20, Short.MAX_VALUE))
        );

        Gravar.setText("OK");
        Gravar.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                OKActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addContainerGap())
                .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                    .addComponent(Gravar)
                    .addGap(230, 230, 230))))
        );
        layout.setVerticalGroup(
```

```
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 43,
Short.MAX_VALUE)
                .addComponent(Gravar)
                .addGap(25, 25, 25))
        );

        java.awt.Dimension screenSize = java.awt.Toolkit.getDefaultToolkit().getScreenSize();
        setBounds((screenSize.width-539)/2, (screenSize.height-407)/2, 539, 407);
    }// </editor-fold>
    public int trataSQL(String recebe) {
        int aux = 0;
        if (recebe.contains("?")) {
            aux = 5;
        } else if (recebe.contains("insert")) {
            aux = 1;
        } else if (recebe.contains("update")) {
            aux = 2;
        } else if (recebe.contains("select")) {
            aux = 3;
        } else if (recebe.contains("delete")) {
            aux = 4;
        }
        return aux;
    }

    public void acaoBotao() {
        Gravar.setEnabled(false);
        int vezes = Integer.parseInt(getVezes());
```

```
boolean autocomm = commitSel.isSelected(); // pega estado do botao autocommit OFF
conDB.getConexao(bdfield.getText(), userfield.getText(), passfield.getText());
if (trataSQL(getSql()) == 1 && statement.isSelected() && vezes != 0) //Verifica se é
INSERT
{
   conDB.insertSqlStat(autocomm, vezes, getSql());
}

if (trataSQL(getSql()) == 1 && prestatement.isSelected() && vezes != 0) //Verifica se é
INSERT
{
   conDB.insertSql(autocomm, vezes, getSql());
}

if (trataSQL(getSql()) == 3 && statement.isSelected() && vezes != 0) //Verifica se é
INSERT
{
   conDB.selectSqlSta(vezes, getSql());
}
if (trataSQL(getSql()) == 3 && prestatement.isSelected() && vezes != 0) //Verifica se é
INSERT
{
   conDB.selectSql(vezes, getSql());
}

if ((trataSQL(getSql()) == 5 || trataSQL(getSql()) == 2 || trataSQL(getSql()) == 4) &&
statement.isSelected()) {
   conDB.execincrSqlStat(autocomm, vezes, getSql());
}
if ((trataSQL(getSql()) == 5 || trataSQL(getSql()) == 2 || trataSQL(getSql()) == 4)  &&
prestatement.isSelected()) {
   conDB.execincrSql(autocomm, vezes, getSql());
}
```

```
        choose.select(0);
        commitSel.setSelected(false);
        prestatement.setSelected(false);
        statement.setSelected(false);
        Gravar.setEnabled(true);
    }
    //método de evento do botão Gravar
    private void OKActionPerformed(java.awt.event.ActionEvent evt) {

        acaoBotao();

}

    //método de evento do radiobutton statement
    private void statementActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        prestatement.setSelected(false);
    }

    //método de evento do radiobutton prestatement
    private void prestatementActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        statement.setSelected(false);
    }


    //método main()
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {

            public void run() {
                new Interface().setVisible(true);
            }
```

```
        });
    }
    // Variables declaration - do not modify
    private javax.swing.JButton Gravar;
    private javax.swing.JTextArea TextSQL;
    private javax.swing.JTextField bdfield;
    private java.awt.Choice choose;
    private javax.swing.JCheckBox commitSel;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextField passfield;
    private javax.swing.JRadioButton prestatement;
    private javax.swing.JRadioButton statement;
    private javax.swing.JTextField userfield;
    // End of variables declaration
}
```

Classe StopWatch

```
package tcc;

/**
 *
 * @author Diego
 */
class StopWatch {

    private double startTime = 0;
    private double stopTime = 0;
    private boolean running = false;

    //método que pega o tempo corrente
```

```
    public void start() {
        this.startTime = System.currentTimeMillis();
        this.running = true;
    }

    //método que pega o tempo corrente
    public void stop() {
        this.stopTime = System.currentTimeMillis();
        this.running = false;
    }

    //tempo decorrido em milisegundos
    public double getElapsedTime() {
        double elapsed;
        if (running) {
            elapsed = (System.currentTimeMillis() - startTime);
        } else {
            elapsed = (stopTime - startTime);
        }
        return elapsed;
    }

    //tempo decorrido em segundos
    public double getElapsedTimeSecs() {
        double elapsed;
        if (running) {
            elapsed = ((System.currentTimeMillis() - startTime) / 1000);
        } else {
            elapsed = ((stopTime - startTime) / 1000);
        }
        return elapsed;
    }
}
```

APÊNDICE B – APLICAÇÃO .NET

Neste apêndice é apresentado o código das classes implementadas na aplicação .NET, utilizadas na execução das avaliações realizadas neste trabalho, sendo apresentados apenas os métodos sem as descrições, pois foram descritos anteriormente.

Classe ConexãoDB

```
package TCC_NET;
import System.Data.OracleClient.*;
import System.Data.OleDb.*;
import java.util.*;
import java.sql.*;
import java.text.*;
import java.io.*;
import System.Windows.Forms.*;
import System.Diagnostics.*;
import System.*;
import System.Data.*;
import System.Data.Odbc.*;


/**
* Summary description for ConexaoDB.
*/
public class ConexaoDB
{
	private OdbcConnection con;
	private OleDbConnection conn;
	private Stopwatch tempo = new Stopwatch();

	public ConexaoDB()
	{
```

```
//
// Construtor
//
}


public OdbcConnection getConexaoODBC(String bd, String user, String pass)
{

        con = new OdbcConnection("Dsn=" + bd + ";Uid=" + user + ";Pwd=" + pass
        + "");
        con.Open();
        con.Close();
        return con;

}

public OleDbConnection getConexaoOleDB(String bd, String user, String pass)
{
        conn = new OleDbConnection("Provider=MSDAORA;Data Source="+
        ";user id="+user+";password="+pass+";"
        + "persist security info=false;");

        conn.Open();
        conn.Close();
        return conn;

}

//Driver ODBC da Oracle
public void insertSqlODBC(boolean autocommit, String vezes, String sql)
{
        OdbcTransaction auto = null;
        tempo.Reset();
```

```
        tempo.Start();
        OdbcCommand save = con.CreateCommand();
        save.set_CommandText(sql);
        con.Open();
        if (autocommit == true)
        {
                auto = con.BeginTransaction();
                save.set_Transaction(auto);
        }

        for (int i = 1; i <= Integer.parseInt(vezes); i++)
        {
                save.ExecuteNonQuery();
        }

        if (autocommit == true)
        {
                auto.Commit();
        }
        con.Close();
        tempo.Stop();
        MessageBox.Show(String.valueOf("Tempo de inserção ODBC foi de: " +
        tempo.get_ElapsedMilliseconds() * 0.001));

}

// Métodos INSERT DotNet

public void insertSqlOleDb(boolean autocommit, String vezes, String sql)
{
        OleDbTransaction auto = null;
        tempo.Reset();
        tempo.Start();
        OleDbCommand save = new OleDbCommand(sql, conn); ;
```

```
        conn.Open();
        if (autocommit == true)
        {
                auto = conn.BeginTransaction();
                save.set_Transaction(auto);
        }
        for (int i = 1; i <= Integer.parseInt(vezes); i++)
        {
                save.ExecuteNonQuery();
        }
        if (autocommit == true)
        {
                auto.Commit();
        }
        conn.Close();
        tempo.Stop();
        MessageBox.Show(String.valueOf("Tempo de inserção OleDB foi de: " +
        tempo.get_ElapsedMilliseconds() * 0.001));

}


public void execincrSqlODBC(boolean autocommit, String vezes, String sql)
{
        OdbcTransaction auto = null;
        tempo.Reset();
        tempo.Start();
        OdbcCommand save = con.CreateCommand();
        con.Open();
        if (autocommit == true)
        {
                auto = con.BeginTransaction();
                save.set_Transaction(auto);
        }
```

```
        for (int i = 1; i < Integer.parseInt(vezes); i++)
        {

                save.set_CommandText(sql.Replace("?", String.valueOf(i)));
                save.ExecuteNonQuery();

        }
        if (autocommit == true)
        {
                auto.Commit();
        }
        con.Close();
        tempo.Stop();
        MessageBox.Show(String.valueOf("Tempo de execução ODBC Incr foi de: "
        + tempo.get_ElapsedMilliseconds() * 0.001));

}

//Driver OleDb da Microsoft
public void execincrOleDb(boolean autocommit, String vezes, String sql)
{
        OleDbTransaction auto = null;
        tempo.Reset();
        tempo.Start();
        OleDbCommand save = conn.CreateCommand();
        conn.Open();
        if (autocommit == true)
        {
                auto = conn.BeginTransaction();
                save.set_Transaction(auto);
        }

        for (int i = 1; i < Integer.parseInt(vezes); i++)
        {
```

```
                save.set_CommandText(sql.Replace("?", String.valueOf(i)));
                save.ExecuteNonQuery();
        }

        if (autocommit == true)
        {
                auto.Commit();
        }
        conn.Close();
        tempo.Stop();
        MessageBox.Show(String.valueOf("Tempo de execução OleDB Incr foi de: "
        + tempo.get_ElapsedMilliseconds() * 0.001));

}

// Fim Métodos INSERT DotNet


// Métodos SELECT DotNet
public void selectSqlODBC(String vezes, String sql)
{

        OdbcCommand sel = new OdbcCommand(sql, con);
        OdbcDataAdapter adapter = new OdbcDataAdapter();
        DataSet dataset = new DataSet();
        tempo.Reset();
        tempo.Start();
        con.Open();
        for (int i = 1; i < Integer.parseInt(vezes); i++)
        {

                sel.set_CommandText(sql.Replace("!", String.valueOf(i)));
                adapter.set_SelectCommand(sel);
                adapter.Fill(dataset);
```

```
}
con.Close();
tempo.Stop();
        MessageBox.Show(String.valueOf("Tempo de recuperação ODBC foi
        de: " + tempo.get_ElapsedMilliseconds() * 0.001));
}

public void selectSqlOleDb(String vezes, String sql)
{
        OleDbCommand sel = new OleDbCommand(sql, conn); ;
        OleDbDataAdapter adapter = new OleDbDataAdapter();
        DataSet dataset = new DataSet();
        tempo.Reset();
        tempo.Start();
        conn.Open();
        for (int i = 1; i < Integer.parseInt(vezes); i++)
        {

                sel.set_CommandText(sql.Replace("!", String.valueOf(i)));
                adapter.set_SelectCommand(sel);
                adapter.Fill(dataset);

        }
conn.Close();
tempo.Stop();
MessageBox.Show(String.valueOf("Tempo de recuperação OleDB foi de: " +
tempo.get_ElapsedMilliseconds() * 0.001));

}

// Fim Métodos SELECT DotNet
```

```
}
```

Classe Interface

```
package TCC_NET;

import System.Collections.Generic.*;
import System.Data.*;
import System.Drawing.*;
import System.ComponentModel.*;
import System.Windows.Forms.*;
import java.sql.*;

/**
 * Summary description for Form1.
 */
public class Interface extends System.Windows.Forms.Form
{
        ConexaoDB conDB = new ConexaoDB();
        private Button gravarButton;
        private Panel Panel;
        private System.ComponentModel.IContainer components;
        private ComboBox escolhaCombo;
        private String sql;
        private RichTextBox TextArea;
        private TextBox bdfield;
        private TextBox passfield;
        private TextBox userfield;
        private Label label3;
        private Label label2;
        private Label label1;
        private CheckBox commitSel;
        private RadioButton OleDbRadio;
        private RadioButton ODBCRadio;
        private String escolha;
```

```
public Interface()
{
        //
        // Inicializa Componentes
        //
        InitializeComponent();



}

public String getChoose(){
        this.escolha = escolhaCombo.get_SelectedItem().toString();
        return this.escolha;
}
public String getSql()
{
        this.sql = TextArea.get_Text();
        return this.sql;


}

#region Windows Form Designer generated code
/**
* Clean up any resources being used.
*/
protected void Dispose(boolean disposing)
{
        if (disposing)
        {
                if (components != null)
                {
                        components.Dispose();
                }
        }
```

```
        super.Dispose(disposing);
}

/**
 * Required method for Designer support - do not modify
 * the contents of this method with the code editor.
 */
private void InitializeComponent()
{
        this.gravarButton = new System.Windows.Forms.Button();
        this.Panel = new System.Windows.Forms.Panel();
        this.OleDbRadio = new System.Windows.Forms.RadioButton();
        this.ODBCRadio = new System.Windows.Forms.RadioButton();
        this.commitSel = new System.Windows.Forms.CheckBox();
        this.label3 = new System.Windows.Forms.Label();
        this.label2 = new System.Windows.Forms.Label();
        this.label1 = new System.Windows.Forms.Label();
        this.passfield = new System.Windows.Forms.TextBox();
        this.userfield = new System.Windows.Forms.TextBox();
        this.bdfield = new System.Windows.Forms.TextBox();
        this.escolhaCombo = new System.Windows.Forms.ComboBox();
        this.TextArea = new System.Windows.Forms.RichTextBox();
        this.Panel.SuspendLayout();
        this.SuspendLayout();
        //
        // gravarButton
        //
        this.gravarButton.set_Location(new System.Drawing.Point(181, 265));
        this.gravarButton.set_Name("gravarButton");
        this.gravarButton.set_Size(new System.Drawing.Size(75, 23));
        this.gravarButton.set_TabIndex(0);
        this.gravarButton.set_Text("OK");
        this.gravarButton.set_UseVisualStyleBackColor(true);
```

```
            this.gravarButton.add_Click(new
System.EventHandler(this.gravarButton_Click));
            //
            // Panel
            //
            this.Panel.set_BorderStyle(System.Windows.Forms.BorderStyle.Fixed3D);
            this.Panel.get_Controls().Add(this.OleDbRadio);
            this.Panel.get_Controls().Add(this.ODBCRadio);
            this.Panel.get_Controls().Add(this.commitSel);
            this.Panel.get_Controls().Add(this.label3);
            this.Panel.get_Controls().Add(this.label2);
            this.Panel.get_Controls().Add(this.label1);
            this.Panel.get_Controls().Add(this.passfield);
            this.Panel.get_Controls().Add(this.userfield);
            this.Panel.get_Controls().Add(this.bdfield);
            this.Panel.get_Controls().Add(this.escolhaCombo);
            this.Panel.get_Controls().Add(this.TextArea);
            this.Panel.set_Location(new System.Drawing.Point(24, 24));
            this.Panel.set_Name("Panel");
            this.Panel.set_Size(new System.Drawing.Size(385, 217));
            this.Panel.set_TabIndex(2);
            this.Panel.add_Paint(new
System.Windows.Forms.PaintEventHandler(this.panel1_Paint));
            //
            // OleDbRadio
            //
            this.OleDbRadio.set_AutoSize(true);
            this.OleDbRadio.set_Location(new System.Drawing.Point(266, 87));
            this.OleDbRadio.set_Name("OleDbRadio");
            this.OleDbRadio.set_Size(new System.Drawing.Size(56, 17));
            this.OleDbRadio.set_TabIndex(12);
            this.OleDbRadio.set_TabStop(true);
            this.OleDbRadio.set_Text("OleDB");
            this.OleDbRadio.set_UseVisualStyleBackColor(true);
```

```
//
// ODBCRadio
//
this.ODBCRadio.set_AutoSize(true);
this.ODBCRadio.set_Location(new System.Drawing.Point(195, 87));
this.ODBCRadio.set_Name("ODBCRadio");
this.ODBCRadio.set_Size(new System.Drawing.Size(55, 17));
this.ODBCRadio.set_TabIndex(11);
this.ODBCRadio.set_TabStop(true);
this.ODBCRadio.set_Text("ODBC");
this.ODBCRadio.set_UseVisualStyleBackColor(true);
//
// commitSel
//
this.commitSel.set_AutoSize(true);
this.commitSel.set_Location(new System.Drawing.Point(266, 129));
this.commitSel.set_Name("commitSel");
this.commitSel.set_Size(new System.Drawing.Size(105, 17));
this.commitSel.set_TabIndex(10);
this.commitSel.set_Text("AutoCommit OFF");
this.commitSel.set_UseVisualStyleBackColor(true);
//
// label3
//
this.label3.set_AccessibleName("");
this.label3.set_AutoSize(true);
this.label3.set_Location(new System.Drawing.Point(222, 171));
this.label3.set_Name("label3");
this.label3.set_Size(new System.Drawing.Size(38, 13));
this.label3.set_TabIndex(9);
this.label3.set_Text("Senha");
//
// label2
//
```

```
this.label2.set_AccessibleName("");
this.label2.set_AutoSize(true);
this.label2.set_Location(new System.Drawing.Point(3, 168));
this.label2.set_Name("label2");
this.label2.set_Size(new System.Drawing.Size(43, 13));
this.label2.set_TabIndex(8);
this.label2.set_Text("Usuário");
//
// label1
//
this.label1.set_AccessibleName("");
this.label1.set_AutoSize(true);
this.label1.set_Location(new System.Drawing.Point(0, 133));
this.label1.set_Name("label1");
this.label1.set_Size(new System.Drawing.Size(87, 13));
this.label1.set_TabIndex(7);
this.label1.set_Text("Banco de Dados");
//
// passfield
//
this.passfield.set_Location(new System.Drawing.Point(266, 168));
this.passfield.set_Name("passfield");
this.passfield.set_Size(new System.Drawing.Size(100, 20));
this.passfield.set_TabIndex(6);
//
// userfield
//
this.userfield.set_Location(new System.Drawing.Point(97, 168));
this.userfield.set_Name("userfield");
this.userfield.set_Size(new System.Drawing.Size(100, 20));
this.userfield.set_TabIndex(5);
//
// bdfield
//
```

```
        this.bdfield.set_Location(new System.Drawing.Point(97, 130));
        this.bdfield.set_Name("bdfield");
        this.bdfield.set_Size(new System.Drawing.Size(100, 20));
        this.bdfield.set_TabIndex(4);
        //
        // escolhaCombo
        //
        this.escolhaCombo.set_FormattingEnabled(true);
        this.escolhaCombo.get_Items().AddRange(new Object[] {
"1",
"10",
"100",
"1000",
"5000",
"10000"}
        );
        this.escolhaCombo.set_Location(new System.Drawing.Point(6, 87));
        this.escolhaCombo.set_Name("escolhaCombo");
        this.escolhaCombo.set_Size(new System.Drawing.Size(106, 21));
        this.escolhaCombo.set_TabIndex(3);
        //
        // TextArea
        //
        this.TextArea.set_Location(new System.Drawing.Point(3, 3));
        this.TextArea.set_Name("TextArea");
        this.TextArea.set_Size(new System.Drawing.Size(375, 66));
        this.TextArea.set_TabIndex(1);
        this.TextArea.set_Text("");
        //
        // Interface
        //
        this.set_AutoScaleDimensions(new System.Drawing.SizeF(6F, 13F));
        this.set_AutoScaleMode(System.Windows.Forms.AutoScaleMode.Font);
        this.set_ClientSize(new System.Drawing.Size(436, 323));
```

```
            this.get_Controls().Add(this.Panel);
            this.get_Controls().Add(this.gravarButton);
            this.set_Name("Interface");
            this.set_Text("TCC");
            this.add_Load(new System.EventHandler(this.Interface_Load));
            this.Panel.ResumeLayout(false);
            this.Panel.PerformLayout();
            this.ResumeLayout(false);

        }
        #endregion
        public void CleanRadios()
        {
            OleDbRadio.set_Checked(false);
            ODBCRadio.set_Checked(false);
        }

        public String TipoCon()
        {
            String tipocon = null;

            if (ODBCRadio.get_Checked())
                tipocon = ODBCRadio.get_Text();
            if (OleDbRadio.get_Checked())
                tipocon = OleDbRadio.get_Text();
            if (OleDbRadio.get_Checked() == false && ODBCRadio.get_Checked() ==
false)
                tipocon = "no";

            return tipocon;
        }

        public int trataSQL(String recebe)
        {
```

```
            int aux = 0;
            if (recebe.Contains("?"))
                  aux = 5;
            else if (recebe.Contains("insert"))
            {
                  aux = 1;
            }
            else if (recebe.Contains("update"))
            {
                  aux = 2;
            }
            else if (recebe.Contains("select"))
            {
                  aux = 3;
            }
            else if (recebe.Contains("delete"))
            {
                  aux = 4;
            }

            return aux;
      }

      public void acaoBotao()
      {
            boolean autocomm = commitSel.get_Checked();
            gravarButton.set_Enabled(false);
            conDB.getConexaoOleDB(bdfield.get_Text(), userfield.get_Text(),
passfield.get_Text());
            conDB.getConexaoODBC(bdfield.get_Text(), userfield.get_Text(),
passfield.get_Text());

            if ((trataSQL(getSql()) == 1) && TipoCon().equals("ODBC"))
                  conDB.insertSqlODBC(autocomm, getChoose(), getSql());
```

```
            else if (trataSQL(getSql()) == 1 && TipoCon().equals("OleDB"))
                conDB.insertSqlOleDb(autocomm, getChoose(), getSql());

            else if (trataSQL(getSql()) == 3 && TipoCon().equals("ODBC"))
                conDB.selectSqlODBC(getChoose(), getSql());

            else if (trataSQL(getSql()) == 3 && TipoCon().equals("OleDB"))
                conDB.selectSqlOleDb(getChoose(), getSql());


            else if ((trataSQL(getSql()) == 5 || trataSQL(getSql()) == 2 ||
trataSQL(getSql()) == 4) && TipoCon().equals("ODBC"))
                conDB.execincrSqlODBC(autocomm, getChoose(), getSql());

            else if ((trataSQL(getSql()) == 5 || trataSQL(getSql()) == 2 ||
trataSQL(getSql()) == 4) && TipoCon().equals("OleDB"))
                conDB.execincrOleDb(autocomm, getChoose(), getSql());



            CleanRadios();
            commitSel.set_Checked(false);
            gravarButton.set_Enabled(true);

        }

        private void gravarButton_Click(Object sender, System.EventArgs e) throws
SQLException
        {
            acaoBotao();

        }
```

```
        private void deletarButton_Click(Object sender, System.EventArgs e)
        {

                trataSQL(getSql());

        }

        private void panel1_Paint(Object sender, PaintEventArgs e)
        {

        }

        private void Interface_Load(Object sender, System.EventArgs e)
        {

        }

}
```