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

**VINÍCIUS SEIXAS TANURE**

# AVALIAÇÃO DE PERFORMANCE DE UM SISTEMA ORIENTADO A OBJETOS UTILIZANDO BDOO

MARÍLIA
2008

VINÍCIUS SEIXAS TANURE

# AVALIAÇÃO DE PERFORMANCE DE UM SISTEMA ORIENTADO A OBJETOS UTILIZANDO BDOO

Trabalho de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, como requisito parcial para obtenção do grau de Bacharelado em Ciência da Computação.

Orientador:
Prof. Dr. Edmundo Sérgio Spoto.

MARÍLIA
2008

TANURE, Vinícius Seixas
Avaliação de performance de um sistema orientado a objetos utilizando BDOO/ Vinícius Seixas Tanure; orientador: Prof. Dr. Edmundo Sérgio Spoto. Marília, SP: [s.n.], 2008.
57 f.

Trabalho de Curso (Graduação em Ciência da Computação) – Curso de Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, Marília, 2008.

1. Avaliação de performance 2. Banco de dados orientado a objetos 3. Java

CDD: 005.74

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
*BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO*

**TRABALHO DE CONCLUSÃO DE CURSO – AVALIAÇÃO FINAL**

Vinícius Seixas Tanure

AVALIAÇÃO DE PERFORMANCE DE UM SISTEMA ORIENTADO A OBJETOS UTILIZANDO BDOO

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: 7,0 ( Sete )

Orientador: Edmundo Sérgio Spoto

1º. Examinador: Paulo Augusto Nardi

2º. Examinador: Fabio Lucio Meira

Marília, 13 de novembro de 2008.

AGRADECIMENTOS

A **Deus**, por tornar nossas conquistas possíveis.

À minha mãe, **Maria Fátima Seixas Cheque de Campos Tanure**, pelo apoio e compreensão.

Ao meu orientador, **Prof. Dr. Edmundo Sérgio Spoto**, pelas críticas e sugestões que nortearam esta monografia.

Aos meus colegas e sócios, **Diego Roberto Colombo Dias, Edmilton Oséias da Cunha, Rafael Serapilha Durelli e Rafael Paes**, pela colaboração e incentivo constantes.

TANURE, Vinícius Seixas. **Avaliação de performance de um sistema orientado a objetos utilizando BDOO**. 2008. 57 f. Trabalho de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2008.

RESUMO

Monografia pertinente à realização de avaliação de performance visando comparar o desempenho de duas versões de um BDOO e sua respectiva aplicação em JAVA: uma versão dispondo de métodos implementados dentro do BDOO (Versão 1) e outra versão dispondo desses mesmos métodos implementados na aplicação JAVA (Versão 2). Para avaliar o desempenho das duas versões foram utilizados os seguintes materiais: uma máquina *Intel Core 2 Duo* 1.8 *Gigahertz*, 1 *Gigabyte* de memória RAM e 120 *Gigabytes* de HD; o SGBDOR *Oracle* 9i; plataforma JAVA *NetBeans* 6; banco de dados anteriormente desenvolvido por Ruiz e Gonçalves (2007). Com quatro variações determinadas do volume de inserção de dados (10, 100, 1.000 e 10.000), ambas as versões foram submetidas a avaliações idênticas para definição do tempo despendido por operação solicitada. Cada uma das operações foi repetida cinco vezes, ficando adotado como resultado do tempo despendido a média aritmética apurada entre as cinco repetições.

**Palavras-chave:** BDOO. Banco de dados. Avaliação de Performance. Performance. Java.

# LISTA DE ILUSTRAÇÕES

Figura 1 – Esquema genérico de compilação de programas JAVA ....................................... 13
Figura 2 – Esquema de relações de herança entre classes ...................................................... 14
Figura 3 – Processo de compilação e interpretação JAVA .................................................... 16
Figura 4 – Exemplo de chamada de procedimento CallableStatement .................................. 26
Figura 5 – Exemplo de envio de comando PreparedStatement .............................................. 27
Figura 6 – Classe StopWatch .................................................................................................. 28
Figura 7 – Exemplo de inserção de dados ................................................................................ 29
Figura 8 – Procedimento de inserção com stored procedure (Versão 1) ................................. 30
Figura 9 – Procedimento de remoção com stored procedure (Versão 1) ................................ 31
Figura 10 – Procedimento de remoção implementado em Java (Versão 2) ............................ 32
Figura 11 – Procedimento de recuperação com stored procedure (Versão 1) ......................... 33
Figura 12 – Procedimento de recuperação implementado em Java (Versão 2) ....................... 34
Figura 13 – Procedimento de comparação com stored procedure (Versão 1) ........................ 35
Figura 14 – Procedimento de comparação implementado em Java (Versão 2) ....................... 36
Figura 15 – Procedimento de atualização com stored procedure (Versão 1) .......................... 37
Figura 16 – Procedimento de atualização implementado em Java (Versão 2) ........................ 38

LISTA DE TABELAS

Tabela 1 – Resultados obtidos no procedimento de inserção (Versão 1)................................39
Tabela 2 – Resultados obtidos no procedimento de remoção (Versão 1) ...............................40
Tabela 3 – Resultados obtidos no procedimento de recuperação (Versão 1)...........................41
Tabela 4 – Resultados obtidos no procedimento de alteração (Versão 1)...............................42
Tabela 5 – Resultados obtidos no procedimento de comparação (Versão 1)...........................43
Tabela 6 – Resultados obtidos no procedimento de inserção (Versão 2)................................45
Tabela 7 – Resultados obtidos no procedimento de remoção (Versão 2) ...............................46
Tabela 8 – Resultados obtidos no procedimento de recuperação (Versão 2)...........................47
Tabela 9 – Resultados obtidos no procedimento de alteração (Versão 2)...............................48
Tabela 10 – Resultados obtidos no procedimento de comparação (Versão2)..........................49

# LISTA DE ABREVIATURAS E SIGLAS

API: *Application Programming Interface* (Interface de Programação de Aplicativos)

BDOO: Banco de Dados Orientado a Objetos

CAD: *Computer Aided Design* (Projeto Auxiliado por Computador)

CASE: *Computer Aided Software Engineering* (Engenharia de Software Auxiliada por Computador)

JDBC: *Java Database Connectivity* (Conexão Java para Banco de Dados)

JVM: *Java Virtual Machine* (Máquina Virtual Java)

LPOO: Linguagem de Programação Orientada a Objetos

MS: Milissegundos

OID: *Object Indentifier* (Identificador de Objeto)

OO: Orientação a Objeto

SGBD: Sistema de Gerenciamento de Banco de Dados

# LISTA DE GRÁFICOS

Gráfico 1 – Comportamento do procedimento de inserção (Versão 1)...................................40
Gráfico 2 – Comportamento do procedimento de remoção (Versão 1)...................................41
Gráfico 3 – Comportamento do procedimento de recuperação (Versão 1).............................42
Gráfico 4 – Comportamento do procedimento de alteração (Versão 1)..................................43
Gráfico 5 – Comportamento do procedimento de comparação (Versão 1).............................44
Gráfico 6 – Comportamento do procedimento de inserção (Versão 2)...................................45
Gráfico 7 – Comportamento do procedimento de remoção (Versão 2)...................................46
Gráfico 8 – Comportamento do procedimento de recuperação (Versão 2).............................47
Gráfico 9 – Comportamento do procedimento de alteração (Versão 2)..................................48
Gráfico 10 – Comportamento do procedimento de comparação (Versão 2)...........................49
Gráfico 11 – Procedimento de inserção (Versão 1 x Versão 2)..............................................50
Gráfico 12 – Procedimento de recuperação (Versão 1 x Versão 2)........................................50
Gráfico 13 – Procedimento de comparação (Versão 1 x Versão 2)........................................51
Gráfico 14 – Procedimento de remoção (Versão 1 x Versão 2).............................................52
Gráfico 15 – Procedimento de alteração (Versão 1 x Versão 2).............................................52
Gráfico 16 – Versão 1 x Versão 2: Comparação de performance (recuperação)....................53
Gráfico 17 – Versão 1 x Versão 2: Comparação de performance (inserção, comparação, remoção e alteração)..........................................................................................................54

# SUMÁRIO

INTRODUÇÃO .......... 10
1 JAVA: LINGUAGEM ORIENTADA A OBJETOS .......... 12
1.1 Definição e Conceitos .......... 12
1.2 APIs e Composição .......... 15
1.3 Exemplo de Aplicação .......... 17
2 BANCO DE DADOS ORIENTADO A OBJETOS .......... 19
2.1 Histórico .......... 19
2.2 Orientação a Objetos e SGBD que aceitam BDOO .......... 20
2.2.1 Objetos .......... 20
2.2.2 Objetos Complexos .......... 21
2.2.3 Hierarquia de Classes .......... 22
2.2.4 Herança .......... 22
2.2.5 Encapsulamento .......... 22
2.2.6 Persistência .......... 23
3 AVALIAÇÃO DE PERFORMANCE: GERAÇÃO E EXECUÇÃO .......... 24
3.1 Geração da Avaliação de Performance da Aplicação de BDOO .......... 24
3.2 Execução da Avaliação de Performance .......... 25
4 RESULTADOS OBTIDOS .......... 39
4.1 Resultados Obtidos na Avaliação da Versão 1 da Aplicação de BDOO .......... 39
4.2 Resultados Obtidos na Avaliação da Versão 2 da Aplicação de BDOO .......... 44
4.3 Análise e Comparação dos Resultados .......... 49
CONCLUSÃO .......... 55
REFERÊNCIAS .......... 56

# INTRODUÇÃO

Na atualidade, bancos de dados são de utilizações necessárias nas mais diversas áreas de atuação humana, sendo apropriados para armazenamento e recuperação de quaisquer tipos de informações, independentemente do volume dessas informações. Algumas áreas desse atendimento irrestrito, contudo, só puderam ter suas informações lançadas em bancos de dados após o desenvolvimento dos sistemas de gerenciamento de bancos de dados que permitem a implementação dos paradigmas de orientação a objetos.

As limitações apresentadas pelos sistemas gerenciadores de bancos de dados (SGBD) convencionais quanto às aplicações em áreas que envolvem tipos complexos de dados, como CAD (*Computer Aided Design*), CASE (*Computer Aided Software Engineering*), Inteligência Artificial, Bancos de Dados de hipertexto, entre outras, motivaram as pesquisas e o desenvolvimento de SGBD, que incorporaram a habilidade para criar aqueles tipos complexos de dados das linguagens de programação orientadas a objetos.

Dentre as linguagens orientadas a objetos que se prestam para operacionalizar os recursos disponibilizados por um SGBD, destaca-se JAVA, a qual, por suas características, tornou-se uma das linguagens de programação mais utilizadas no planeta (MENGUE, 2002), sendo também a adotada para efeito de implementação da aplicação necessária ao teste de performance realizado neste trabalho.

Por ser um sistema jovem, com primeira aparição no início da década de 90, esse tipo de SGBD mostra-se ainda um campo vasto para pesquisas, a despeito dos inúmeros trabalhos científicos que sobre ele são cotidianamente publicados.

Dentro desse vasto campo de pesquisas, foi escolhida, para enfrentamento neste trabalho, a questão relacionada à comparação da performance de duas versões de um BDOO e sua respectiva aplicação: uma versão dispondo de métodos implementados dentro do BDOO e outra versão dispondo desses mesmos métodos implementados na aplicação JAVA.

O objetivo deste trabalho, portanto, foi verificar se as duas versões testadas apresentam diferença de performance, bem como, em caso positivo, estabelecer qual delas apresenta melhor desempenho para cada um dos casos propostos.

Os materiais utilizados para realização dos testes foram: uma máquina *Intel Core 2 Duo* 1.8 *Gigahertz*, 1 *Gigabyte* de memória RAM e 120 *Gigabytes* de HD; o SGBDOR *Oracle* 9i; plataforma JAVA *NetBeans* 6; banco de dados anteriormente desenvolvido por Ruiz e Gonçalves (2007); e classe *StopWatch*.

A metodologia utilizada neste trabalho consistiu em: instalação das ferramentas *Oracle* 9i e *NetBeans* 6 no computador destinado à pesquisa; execução dos scripts de criação do banco de dados desenvolvido por Ruiz e Gonçalves (2007) na ferramenta *Oracle* 9i; desenvolvimento, na Ferramenta *NetBeans* 6, de uma aplicação JAVA para banco de dados em duas versões – uma dispondo de métodos implementados dentro do BDOO (Versão 1) e outra dispondo dos mesmos métodos implementados na aplicação JAVA (Versão 2); adoção da classe *StopWatch* para medição do tempo de execução; execução de testes idênticos em ambas as versões, envolvendo quatro variações determinadas do volume de inserção de dados (10, 100, 1.000 e 10.000); definição e anotação do tempo despendido na execução das operações solicitadas (inserção, recuperação, remoção, atualização e comparação) mediante apuração da média aritmética entre cinco repetições de cada uma das operações; análise e comparação dos resultados obtidos; identificação da melhor performance entre as duas versões, para cada caso de teste.

Com a metodologia acima especificada, o presente trabalho encontra-se organizado em quatro seções seguidas da conclusão. Na seção 1 são apresentados os conceitos básicos, características e aplicação da linguagem orientada a objetos JAVA. Na Seção 2 são descritos os conceitos de orientação a objetos para SGBD e na Seção 3 são abordados os conceitos e definições relativos a teste de performance, declaradas as ações pertinentes à geração do teste de performance das duas versões da aplicação de BDOO, bem como especificadas as condutas de planejamento e execução dos casos de teste. Na Seção 4 são mostrados e analisados os resultados obtidos em todos os testes realizados nas duas versões da aplicação; são também relacionados e analisados, na mesma Seção 4, os resultados baseados no tempo de execução por quantidade de objetos. Após a seção 4 são apresentadas as conclusões finais.

# 1 JAVA: LINGUAGEM ORIENTADA A OBJETOS

No início da década de 90 os laboratórios da Sun Microsystems desenvolveram uma linguagem computacional destinada a ser mais simples e eficiente do que suas antecessoras. Tratava-se da linguagem hoje denominada JAVA, cujo alvo inicial era a produção de software para produtos eletrônicos de consumo (fornos de microondas, agendas eletrônicas, etc.) que requisitam código compacto e arquitetura neutra.

A respeito da trajetória de utilização da linguagem JAVA, assim discorre Indrusiak (1996, p.2):

> A linguagem obteve sucesso em cumprir os requisitos de sua especificação, mas apesar de sua eficiência não conseguiu sucesso comercial. Com a popularização da rede Internet, os pesquisadores da Sun Microsystems perceberam que aquele seria um nicho ideal para aplicar a recém criada linguagem de programação. A partir disso, adaptaram o código Java para que pudesse ser utilizado em microcomputadores conectados à rede Internet, mais especificamente no ambiente da World Wide Web. Java permitiu a criação de programas batizados applets, que trafegam e trocam dados através da Internet e se utilizam da interface gráfica de um web browser. Implementaram também o primeiro browser compatível com a linguagem, o HotJava, que fazia a interface entre as aplicações Java e o sistema operacional dos computadores. Com isso, a linguagem conseguiu uma popularização fora de série, passando a ser usada amplamente na construção de documentos web que permitam maior interatividade.

Atualmente, como salientam Deitel e Deitel (2003), JAVA também é utilizada para desenvolver aplicativos corporativos de grande porte, para aprimorar a funcionalidade de servidores da World Wide Web (os computadores que fornecem o conteúdo que vemos em nossos navegadores da Web), fornecer aplicativos para dispositivos destinados ao consumidor final (como telefones celulares, pagers e assistentes pessoais digitais) e para muitas outras finalidades, impulsionando grandes avanços da computação mundial, como: acesso remoto a banco de dados; bancos de dados distribuídos; comércio eletrônico, interatividade em ambientes de realidade virtual distribuídos; ensino à distância (INDRUSIAK, 1996).

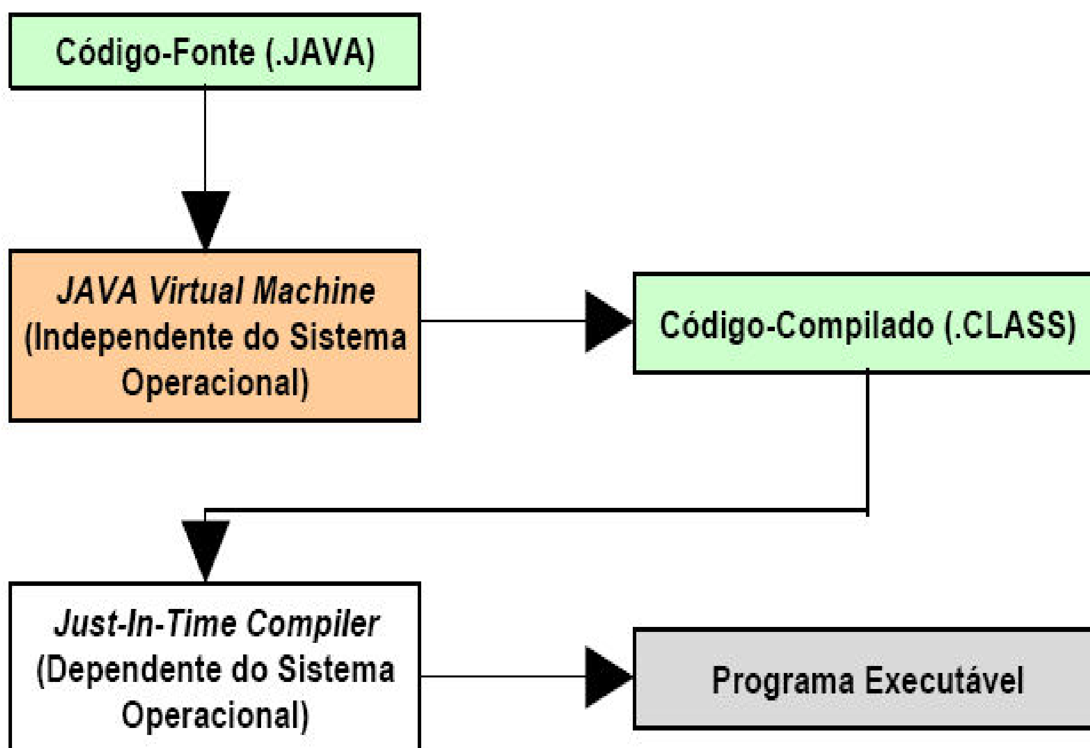
## 1.1 Definição e Conceitos

JAVA é uma linguagem computacional orientada a objetos, adequada para o desenvolvimento de aplicações baseadas na rede Internet, redes fechadas e programas stand-alone (INDRUSIAK, 1996).

A Linguagem JAVA é multiplataforma. Essa característica permite que um programa escrito na Linguagem JAVA possa ser executado em qualquer plataforma (sistema operacional) sem necessidade de alterações no código fonte. Como esclarece Santos Júnior

(2006, p. 3), tal funcionalidade é possível devido à estrutura de linguagem interpretada que caracteriza a linguagem JAVA, e o processo de compilação do código-fonte, conforme ilustrada pela Figura 1:

**Figura 1 – Esquema genérico de compilação de programas JAVA**




Fonte:
http://www.inf.pucpcaldas.br/~joao/cursos/javaxml/MateriaisApoio/ApostilaLinguagemJAVA.pdf

A compreensão de JAVA exige o conhecimento de alguns conceitos básicos e terminologias próprios de linguagem orientada a objetos, como objetos, abstração, atributos, comportamentos, classes, herança, encapsulamento, polimorfismo, que são abordados um a um, a seguir, para melhor esclarecimento do tema, começando pelo termo norteador de toda a linguagem JAVA – *objetos*.

O mundo real, do qual participamos, é composto por inúmeros *objetos* – mesas, cadeiras, carros, motos, animais, pessoas, etc. Através da habilidade da *abstração* nós somos capazes de visualizar mentalmente determinado objeto e considerar isoladamente um ou mais elementos do todo. Podemos pensar que mesas e cadeiras são compostas por pés e tampos, e também que mesas e cadeiras compõem restaurantes; podemos pensar que carros e motos são compostos por rodas e faróis, e também que carros e motos compõem estabelecimentos revendedores de veículos. Pode-se verificar que todos os objetos, quaisquer que sejam eles,

"têm *atributos*, como tamanho, forma, cor e peso e todos eles exibem *comportamentos* (p. ex.: a bola rola, salta, incha e esvazia; o bebê chora, dorme, engatinha, caminha e pisca; o carro acelera, freia e muda de direção; a toalha absorve água)" (DEITEL; DEITEL, 2003, p. 67).

Objetos diferentes entre si podem ter *atributos* e exibir *comportamentos* semelhantes.

Objetos que apresentam mesmas características relativas a *atributos* e *comportamentos* pertencem a uma mesma *classe*.

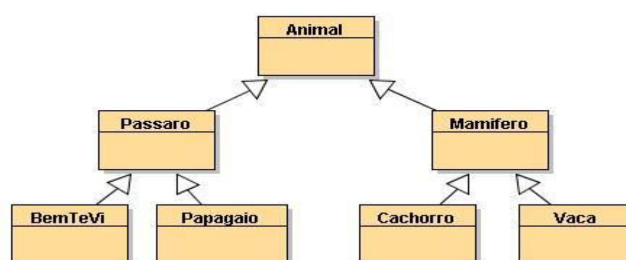
Em JAVA, a *classe* abriga *componentes* que definem o modelo, a receita para criar um objeto. Nela estão contidos *métodos* (que implementam os comportamentos da classe) e *atributos* (que implementam os dados da classe). Como destacam Deitel e Deitel (2003, p. 68):

> Cada classe contém dados e um conjunto de funções que manipulam aqueles dados. Os componentes de dados de uma classe são conhecidos como *atributos*. Os componentes funções de uma classe são conhecidos como *métodos*. Da mesma maneira que uma instância de um tipo primitivo da linguagem, como **int**, é chamada de *variável*, uma instância de um tipo definido pelo usuário (i.e., uma classe) é chamada de *objeto*. O programador usa tipos primitivos como blocos de construção para construir tipos definidos pelo usuário. O foco de atenção em Java está nas classes (com as quais criamos objetos) e não nas funções.

Segundo Martin et al. (1995, p. 20), *encapsulamento* é o ato de empacotar ao mesmo tempo dados e métodos, de forma que o "objeto esconde seus dados de outros objetos e permite que os dados sejam acessados por intermédio de seus próprios métodos", a que se chama ocultação de informações (*information hiding*).

Quando uma nova classe a ser criada possui as mesmas características de outra já existente, pode-se utilizar a relação de *herança*. A relação de *herança* permite o aproveitamento das características já contidas na classe existente, às quais são então adicionadas as características próprias da nova classe. A nova classe, beneficiária da *herança,* é denominada *subclasse* e a classe anteriormente existente, autora da herança, é denominada *superclasse*. A Figura 2 exemplifica relações de *herança* entre indivíduos (*objetos*) do reino animal:

**Figura 2 – Esquema de relações de herança entre classes**




Fonte: http://www.inf.pucrs.br/~flash/lapro2/aula_heranca/

Como é visualizado na Figura 2, subclasses são versões mais específicas das superclasses das quais derivam. O objeto **Animal** é dotado da característica genérica "emitir som", herdada pelos objetos **BemTeVi**, **Papagaio**, **Cachorro**, **Vaca**, que por sua vez emitem sons específicos (late, muge, etc.). Utilizando JAVA, a mesma mensagem (emitir_som()), enviada através de uma superclasse para vários objetos da hierarquia, assume muitas formas de resultados (latido, mugido, etc.). Esse mecanismo de escolha do método específico sobrescrito na subclasse, que implica em muitas formas de resultados, é denominado *polimorfismo*. Ricarte (2000) define *polimorfismo* como:

> o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação (assinatura) mas comportamentos distintos, especializados para cada classe derivada, usando para tanto uma referência a um objeto do tipo da superclasse. A decisão sobre qual o método que deve ser selecionado, de acordo com o tipo da classe derivada, é tomada em tempo de execução, através do mecanismo de ligação tardia.

Exemplo de *polimorfismo* é dado por Rumbaugh et al. (1994, p. 4):

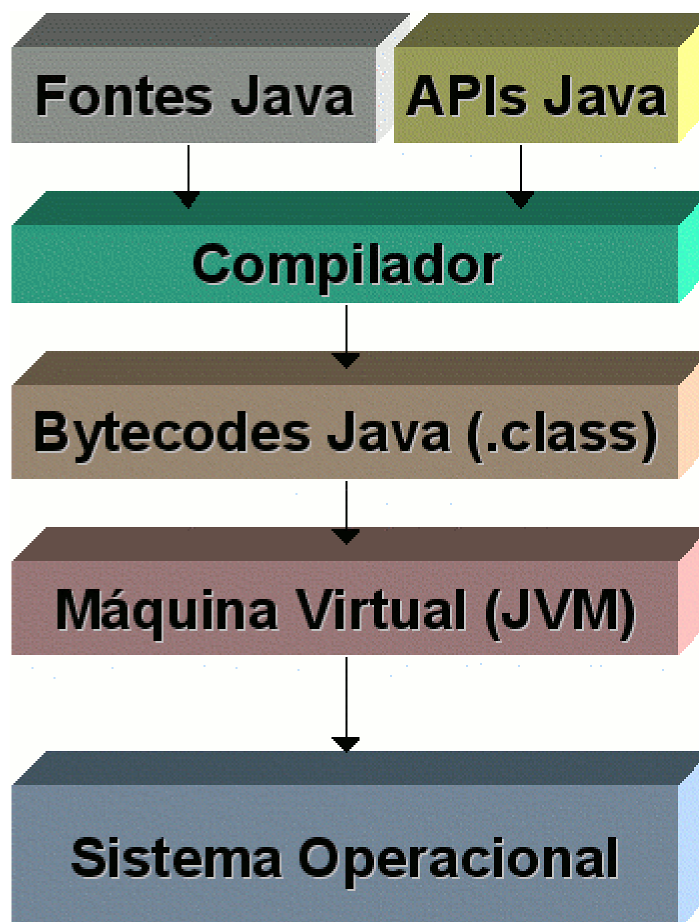
> Polimorfismo significa que a mesma operação pode atuar de modos diversos em classes diferentes. A operação move (mover), por exemplo, pode atuar de forma diferente nas classes Janela e PecadeXadrez.

Dentre os conceitos abordados neste tópico, Deitel e Deitel (2003) destacam que a *abstração* de dados, a *herança* e o *polimorfismo* são os pontos cruciais da programação orientada a objetos, a qual, uma vez implementada em linguagem JAVA, não utiliza ponteiros, *templates* ou herança múltipla. Os programas JAVA são compilados para *bytecodes* que são interpretados por uma máquina virtual (*Java Virtual Machine* ou JVM), sendo a portabilidade alcançada a partir do momento em que máquinas virtuais JAVA estejam disponíveis em diversas plataformas (RICARTE, 1998).

## 1.2 APIs e Composição

Plataformas muito utilizadas para execução de um programa, como o Windows, o Linux, o MacOS, envolvem um conjunto de hardware e software. A plataforma JAVA não envolve hardware; ela utiliza a plataforma de hardware das outras e conta com dois componentes (software) que dão suporte à execução e à construção do programa – a *Java Virtual Machine* (Java VM ou JVM) e a *Java Application Programming Interface* (Java API) –, visualizados no esquema da Figura 3:

**Figura 3 – Processo de compilação e interpretação JAVA**




Fonte:http://dfm.ffclrp.usp.br/~evandro/ibm1030/intro_java/java_basics.html

A JVM, como referido anteriormente, interpreta os *bytecodes* gerados pelo compilador JAVA. Para que um produto consiga executar programas em JAVA (como um browser que executa applet's) é necessário que ele possua uma cópia da JVM.

A API, que é o elemento de construção de programas sob enfoque nesta seção, "é uma coleção de componentes de software prontos, que incluem desde estruturas para manipulação de arquivos até a construção de aplicativos gráficos" (MENGUE, 2002, p. 2). A organização da API é feita com classes e interfaces reunidas em bibliotecas agrupadas, sendo que estas bibliotecas são chamadas de pacotes.

Um bom conhecimento do conteúdo das APIs disponíveis facilita o trabalho do programador que pode se valer de objetos já existentes, ao invés de ter que implementar todas as funções necessárias para sua criação.

No desenvolvimento de um programa em JAVA, uma maneira de reutilizar classes da API, além da herança, é a *composição*. A *composição* é o procedimento que resulta na implementação de uma classe que tem referências a objetos de outras classes como membros, gerando então um objeto do tipo composto que possui mais do que um objeto dentro dele, conforme esclarecem e exemplificam Deitel e Deitel (2003, p. 402):

> Um objeto da classe **AlarmClock** precisa saber quando ele deve soar o alarme, então porque ele não inclui uma referência a um objeto **Time** como membro do objeto **AlarmClock**? Essa capacidade se chama *composição*. A classe pode ter referências a objetos de outras classes como membros.

Em suma, a linguagem JAVA permite que o programador, além de criar suas próprias classes, também se utilize das muitas classes prontas contidas nas extensas bibliotecas das APIs JAVA, proporcionando-lhe inúmeras possibilidades de aplicações.

## 1.3 Exemplo de Aplicação

Dentre as várias aplicações da linguagem JAVA, um exemplo se mostra bastante apropriado para ser inserido no contexto deste trabalho, que tem por enfoque a análise de performance de banco de dados OO – sistema de cadastro.

Genericamente, um sistema de cadastro contém métodos básicos destinados a incluir, excluir e consultar dados armazenados em um banco. Todas essas operações assumem comportamentos diversos dependendo da solicitação do operador (p. ex.: o método *excluir* pode tanto remover todos os objetos de um banco de dados, como pode remover apenas os objetos selecionados segundo a opção do operador). A ativação dos métodos é realizada por meio de uma interface simples, com recursos suficientes para atender as peculiaridades de manipulação dos dados exigidas pelo operador. Para que o banco de dados cumpra o comando do operador e retorne a resposta, o sistema de cadastro também contém métodos responsáveis pela conexão entre a aplicação (o próprio sistema de cadastro) e o SGBD.

O programador incumbido de desenvolver um sistema de cadastro pode gerar por critérios próprios os métodos básicos, a interface, e a conexão acima referidos, ou pode se valer da documentação das APIs JAVA afins, como as ferramentas de apoio SWING e JDBC, relacionadas à interface gráfica e à conexão com o banco, respectivamente. Vale lembrar que, por se tratar de linguagem de acesso livre, as APIs não contidas no pacote básico podem ser obtidas gratuitamente no site da própria Sun Microsystems ou mesmo em sites voltados a desenvolvimento de software.

Com o auxílio das APIs então, o sistema de cadastro, para chegar a ser executado, deve cumprir as cinco fases que, segundo Deitel e Deitel (2003), normalmente se verificam durante o desenvolvimento até a execução de um programa JAVA. São elas: edição, compilação, carga, verificação e execução.

Na Fase 1 (edição), o programador do sistema de cadastro digita o programa em JAVA utilizando um programa editor e fazendo as correções necessárias. Quando o programador solicita que o editor salve o arquivo nele gerado, o programa é armazenado em um dispositivo de armazenamento secundário (DEITEL; DEITEL, 2003, p. 62).

Na Fase 2 (compilação) ocorre a compilação do programa, oportunidade em que o compilador JAVA traduz o código digitado para *bytecodes*.

Na Fase 3 (carga) o programa é colocado na memória principal para que possa ser executado. Essa tarefa é realizada pelo *carregador de classe*, que pega os arquivos que contêm os *bytecodes* e os transfere para a memória.

Na Fase 4 (verificação) os *bytecodes* são verificados pelo *verificador de bytecode*. Essa medida assegura que os *bytecodes* provenientes de classes baixadas da Internet são válidos e não violam as restrições de segurança de JAVA.

Na Fase 5 (execução), finalmente o computador destinatário do sistema de cadastro aciona a JVM, que interpreta o programa, um *bytecode* por vez, realizando então as ações solicitadas pelo operador. A execução se realiza de forma plena e satisfatória desde que não ocorram falhas de fases precedentes, em razão de erros não detectados pelo programador durante o desenvolvimento do *software*.

# 2 BANCO DE DADOS ORIENTADO A OBJETOS

Como esclarece Silberschatz et al. (1999, p. 249), a proposta básica dos sistemas de banco de dados é o gerenciamento de grandes volumes de informação.

Uma referência a *banco de dados* reporta a uma coleção de dados de interesse de uma empresa particular, de uma entidade privada, de uma organização governamental ou do público em geral. O gerenciamento desses dados envolve a definição de estruturas para armazenamento das informações e o fornecimento de mecanismos para manipulá-las. Esse gerenciamento se realiza por meio de um *sistema gerenciador de banco de dados* – o software que permite criar, manter e manipular bancos de dados para diversas aplicações. Um banco de dados criado, mantido e manipulado por um sistema de gerenciamento de banco de dados constitui um *sistema de banco de dados* (RICARTE, 1998).

No caso de banco de dados orientado a objetos, o sistema de gerência, diferentemente dos bancos de dados tradicionais, apresenta as informações organizadas dentro de objetos, não se restringindo somente a tabelas (GONÇALVES; PESENTE; LIMA, [s.d.]).

Em um BDOO, os objetos não incorporam só dados, mas também o comportamento desses dados, incluindo uma relação de todos os módulos do programa que têm atuação sobre eles. Além disso, não há necessidade de se limitar os dados a tipos simples (*number*, *text*, *date*), pois um objeto admite a inclusão de tabelas, desenhos, imagens, voz, música ou combinações de outros objetos, tornando o sistema apropriado para várias áreas não atendidas pelos bancos de dados relacionais.

## 2.1 Histórico

Como anota Sanches (2005), em meados da década de 80 foram detectadas várias áreas, incluindo medicina, multimídia e física de energia elevada, onde bancos de dados relacionais não eram aplicáveis em razão dos tipos de dados envolvidos, os quais exigiam flexibilidade de representação e de acesso. Tiveram início, então, juntamente com o surgimento, na indústria, de linguagens de programação orientadas a objetos, as pesquisas voltadas a bancos de dados orientados a objetos, nos quais os próprios usuários poderiam definir a representação dos dados, bem como os métodos para acessá-los.

O primeiro SGBD com tais características teve sua aparição no início de 1990, por meio da companhia *Objectivity*, permitindo que usuários criassem sistemas de banco de dados

destinados a armazenar resultados de pesquisas como as realizadas pelo CERN (maior laboratório que trabalha com partículas físicas em pesquisas nucleares - europeu) e SLAC (Centro de Aceleração Nuclear - norte-americano), bem como para mapeamento de rede de provedores de telecomunicações e para armazenamento de registros médicos de pacientes em hospitais, consultórios e laboratórios (SANCHES, 2005).

Com a introdução dos bancos de dados orientados a objetos, formou-se a idéia de que estes conquistariam grande parcela do mercado. Posteriormente, porém, verificou-se que os bancos de dados orientados a objetos mostram-se mais apropriados às aplicações especializadas, enquanto os sistemas relacionais continuam a dar sustentação aos negócios tradicionais, nos quais são suficientes as estruturas de dados baseadas em relações (TAKAI; ITALIANO; FERREIRA, 2005).

## 2.2 Orientação a Objetos e SGBD que aceitam BDOO

Os SGBD que aceitam BDOO são o resultado da combinação de idéias aproveitadas dos modelos de dados tradicionais e dos conceitos extraídos de linguagens de programação orientadas a objetos.

Em tais SGBD, a noção de objeto é delineada no nível lógico e adota características próprias não encontradas nas linguagens de programação tradicionais, como operadores de manipulação de estruturas, gerenciamento de armazenamento, tratamento de integridade e persistência dos dados.

Em suma, no desenvolvimento desses SGBD são empregados conceitos utilizados pelas linguagens de programação orientadas a objetos, com as adaptações necessárias. Conceitos de orientação a objetos aplicados em BDOO são abordados em subseções que seguem.

### 2.2.1 Objetos

Em BDOO, define-se objeto como a unidade de informação que contém tanto dados como o comportamento desses dados. Quanto ao conteúdo referente aos dados propriamente ditos, um objeto no BDOO é equivalente a um registro nos bancos de dados convencionais e à uma entidade no modelo entidade-relacionamento.

O modelo de dados orientado a objetos incorpora, das Linguagens de Programação Orientadas a Objetos (LPOO), as características de objetos pertinentes às noções de estrutura de dados e de comportamento (G. F. JÚNIOR et al., 2002 apud DIAS, 1999). Assim, cada ocorrência de um objeto no banco de dados é denominada de instância do objeto.

Um objeto instanciado possui uma identificação. O fato de se mostrar imprópria a identificação dos objetos por meio dos valores de seus atributos ou de sua estrutura comportamental implicou na definição de identificadores únicos, geralmente gerados e administrados pelo próprio SGBD. Além de servir como identificador único de cada objeto, independentemente do valor desse objeto e de seu local físico de seu armazenamento, a identidade também cumpre a tarefa de identificar o objeto como atributo perante os demais objetos, eliminando, dessa forma, anomalias de atualização e de integridade referencial, pois a atualização de um objeto será automaticamente refletida nos objetos que o referenciam, não havendo alteração do valor do identificador do objeto (G. F. JÚNIOR et al., 2002 apud KHOSHAFIAN, 1994; GONÇALVES; PESENTE; LIMA, [s.d.]). A respeito do identificador do objeto (OID: *object indentifier*), Edelweiss e Galante ([s.d.], p. 4) fazem as seguintes colocações:

> Enquanto os objetos possuem OID imutável que os identifica unicamente, os valores não possuem OID, são embutidos nos objetos e não podem ser individualmente referenciados. Assim, um objeto pode ter seu valor atualizado sem perder sua identidade, o que significa que a identidade de um objeto é independente de seu valor.

### 2.2.2 Objetos Complexos

Objetos complexos são formados por construções estruturadas (conjuntos, listas, *tuplas*, coleções, *arrays*) aplicadas a objetos simples (inteiros, booleanos, *strings*) (G. F. JÚNIOR et al., 2002 apud GRAHL et al., 2000). Nos modelos orientados a objetos aquelas estruturas construtoras são em geral ortogonais, ou seja, qualquer delas pode ser aplicada a qualquer objeto. Esses tipos de dados estruturados também são utilizados em BDOO, o que torna mais detalhada a consulta ao banco de dados, pois, "ao invés de acesso a tabelas e registros, é necessário o acesso a listas, tuplas, arrays, etc..." (GONÇALVES; PESENTE; LIMA, [s.d.], p. 2).

### 2.2.3 Hierarquia de Classes

As declarações de classes são armazenadas pelos SGBD como parte do esquema de banco de dados (G. F. JÚNIOR et al., 2002 apud OBJECTSTORE, 1999). Com a criação de uma hierarquia de classes, as alterações da estrutura do banco de dados realizadas por meio de inclusão de novos atributos ou métodos nos objetos tornam-se muito mais flexíveis. O agrupamento hierárquico também possibilita a evolução do esquema do banco de dados mediante a adição de novas classes na hierarquia.

### 2.2.4 Herança

Os SGBD têm a capacidade de promover o gerenciamento do conceito de herança dentro de uma hierarquia de classes armazenáveis. Vale dizer que, tal como nas LPOO, nos SGBD também se pode criar novas classes em função das já existentes. Segundo G. F. Júnior et al. (2000 apud GRAHL et al., 2000), as principais vantagens de se utilizar herança consistem em: uma maior expressividade na modelagem dos dados; maior facilidade de reuso de objetos; possibilidade de definição de classes por refinamento; e inexistência de código redundante.

### 2.2.5 Encapsulamento

Em BDOO, o encapsulamento diz respeito à restrição do conhecimento, por um objeto, do modo como se encontram implementadas as características e comportamento de outro objeto, permitindo que apenas sejam visualizadas externamente as informações que o objeto julgue apropriadas, de acordo com o contexto da aplicação.

Boscarioli et al. ([s.d.]) esclarecem e exemplifica que o conceito de encapsulamento também pode ser utilizado para acoplar dados com operações comumente executadas, como obter a idade de um Veículo. Os mesmos autores mencionam ainda que, se fosse o caso, neste exemplo, seria possível tornar o atributo Ano invisível para os objetos externos e permitir a visualização apenas da operação Idade.

### 2.2.6 Persistência

Os BDOO adotam uma característica que não é apresentada pelas linguagens de programação orientadas a objetos – a persistência dos objetos. Com efeito, enquanto nestas linguagens de programação os objetos existem somente durante a execução do programa, nos BDOO os objetos podem ter sua existência garantida mesmo após o término do programa, mediante armazenamento de seu estado em disco. Segundo G. F. JÚNIOR et al. (2002 apud NASSU; SETZER, 1999), são três as formas pelas quais um sistema torna o objeto persistente:

a) Por classe. Os objetos pertencentes às classes assim declaradas serão persistentes;
b) Por chamada explícita. O objeto tornar-se persistente após a sua criação, através de um comando específico;
c) Por referência. Objetos referenciados por objetos persistentes, tornam-se persistentes.

# 3 AVALIAÇÃO DE PERFORMANCE: GERAÇÃO E EXECUÇÃO

A tarefa de efetuar testes em software, incluindo os direcionados a banco de dados, tem sido objeto de grandes investimentos por parte das empresas do setor. A finalidade de tal medida é reduzir custos com manutenção e gerar um produto de melhor qualidade, tanto para atendimento dos requisitos funcionais (todas as operações que possam ser realizadas pelo sistema, mediante comandos dos usuários ou ainda pela ocorrência de eventos internos ou externos ao sistema) quanto para atendimento dos requisitos não-funcionais (as qualidades globais do software, como usabilidade, segurança e performance).

Dentre os vários tipos de teste, destaca-se o teste de performance, que é foco deste trabalho, sendo utilizado, como acima mencionado, para definir o tempo consumido pelo sistema na realização de suas tarefas em condições diversas.

O teste de performance entra em cena após a constatação de que o sistema em desenvolvimento encontra-se operando adequadamente. Por ser aplicável em sistemas finalizados, o teste de performance se presta também para comparar o desempenho de produtos similares.

O teste de performance, que deve ser executado em ambiente que se aproxime ao máximo do ambiente de produção, envolve a utilização de recursos humanos, como o projetista do teste e o testador de performance, mais os recursos de hardware e de software, com suas respectivas configurações (LEITÃO et al., 2003).

Por denotar a qualificação do tempo de resposta de um software, o teste de performance motiva o pesquisador a buscar sempre uma nova opção que implique na redução daquele tempo. Com isso, o teste de performance é o grande responsável pela constante evolução que hoje se verifica no desempenho dos sistemas computacionais.

## 3.1 Geração da Avaliação de Performance da Aplicação de BDOO

Como já referido em abordagem anterior a respeito de materiais e métodos utilizados neste trabalho, a geração da avaliação de performance destinado a avaliar as duas versões de uma aplicação de banco de dados teve início com a instalação das ferramentas *Oracle* 9i e *NetBeans* 6 no computador destinado à pesquisa. A utilização dessas ferramentas se deu mediante licença do Laboratório de Engenharia de Software (LES) da UNIVEM.

A instalação da ferramenta *Oracle* 9i tornou possível a execução dos *scripts* de criação do banco de dados e a medição dos tempos de resposta verificados durante o teste de performance.

Os scripts de criação do banco de dados foram extraídos do sistema anteriormente desenvolvido por Ruiz e Gonçalves (2007), mais precisamente do módulo relacionado ao setor de locação de equipamentos, que compõe, com outros módulos, um sistema apropriado a uma empresa de Locação de Equipamentos de Construção Civil. Tal sistema abriga os dados de suporte da execução do teste de performance, sendo que esses dados são relacionados a: clientes físicos e jurídicos; equipamentos; preços das locações diárias, semanais e mensais, por equipamento; funcionários envolvidos com as locações.

Após execução dos citados *scripts*, foi promovido, na ferramenta *NetBeans* 6, o desenvolvimento das duas versões da aplicação JAVA destinadas a cumprir as operações pertinentes a inserção, recuperação, remoção, atualização e comparação de dados durante a execução do teste de performance.

Para medição precisa do tempo despendido na execução das operações solicitadas, foi adotada a classe *StopWatch*, que possui métodos apropriados para tanto.

Feita a conexão das versões da aplicação JAVA ao banco de dados, por meio do *driver* JDBC para *Oracle*, e adotada a classe *StopWatch* para medição do tempo, chegou a termo a geração da avaliação comparativa daquelas versões, uma dispondo de métodos implementados em objetos do BDOO (Versão 1) e outra dispondo dos mesmos métodos implementados na aplicação JAVA (Versão 2).

## 3.2 Execução da Avaliação de Performance

Na execução da avaliação de performance, as duas versões da aplicação de BDOO (Versão 1 e Versão 2) foram submetidas a idênticos esforços quantitativos (10, 100, 1.000, 10.000) por operação solicitada (inserção, recuperação, remoção, atualização e comparação).

Na Versão1, as chamadas das operações solicitadas foram realizadas por meio da classe *CallableStatement*, enquanto na Versão 2 o envio de comandos de operações ao banco de dados foi realizado pela classe *PreparedStatement*, como exemplificam as Figuras 4 e 5.

**Figura 4 – Exemplo de chamada de procedimento CallableStatement**

```
1.  public class Insercao1 {
2.  
3.  
4.      public static void main(String... args){
5.          StopWatch s = new StopWatch();
6.          try{
7.  
8.  
9.  
10.             Class.forName("oracle.jdbc.driver.OracleDriver");
11.             Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:vica",
12.                     "scott", "tiger");
13.             System.out.println("deu certo");
14.            CallableStatement pre = conn.prepareCall("{call insere_fisico(?)}");
15.  
16.  
17.  
18.             pre.setInt(1, 10);
19.             s.start();
20.             pre.executeUpdate();
21.             s.stop();
22.             System.out.println("procedure executado");
23.             JOptionPane.showMessageDialog(null,"Tempo de execução"+ s.getElapsedTime());
24.  
25. conn.close();
26.         } catch(SQLException e) {
27.  
28.             e.printStackTrace();
29.  
30.         } catch(ClassNotFoundException e) {
31.  
32.             e.printStackTrace();
33.  
34.         }
35.  
36.     }
37.  
38. }
```

**Figura 5 – Exemplo de envio de comando PreparedStatement**

```
1.  public class Insercao2 {
2.
3.
4.      public static void main(String... args){
5.
6.          StopWatch s = new StopWatch(); // cria objeto da classe cronômetro
7.
8.        try{
9.
10.     Class.forName("oracle.jdbc.driver.OracleDriver");// utilização do driver jdbc para Oracle
11.     Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:vica",
12.             "scott", "tiger");
13.     // cria uma conexão com o banco
14.
15.
16.     System.out.println("deu certo");
17.
18.
19.     // criação do prepared statement para envio de comandos para serem executados no banco
20.     PreparedStatement pre2 = conn.prepareStatement("insert into t_clientefisico select ?," +
21.             "'Gustavo Gonçalves'," +
22.     " to_date('12/12/1986','dd/MM/yyyy'), ref(e), ref(t),'36173107880' from t_endereco e, " +
23.     "t_telefone t where e.codend = 1 and t.codtel = 1  ");
24.
25.           s.start(); // dispara o cronômetro
26.
27.           for (int i=1; i<=10; i++ ) {// laço responsavel pela repetição do numero de operações
28.
29.
30.            pre2.setString(1, Integer.toString(i));//a cada iteração o campo variavel do
31.                                                   // prepared statement é setado com o
32.                                                   // índice do laço
33.
34.            pre2.executeUpdate();// executa o statement
35.
36.           }
37.
38.
39.             s.stop(); //para o cronômetro
40.
41.             System.out.println("procedure executado");
42.
43.
44.             //uso do método getElapsedTime()  para imprimir o tempo em milissegundos
45.             JOptionPane.showMessageDialog(null,"Tempo de execução"+ s.getElapsedTime());//
46.
47.       conn.close(); //fecha a conexão com o banco
48.
49.         } catch(SQLException e) {
50.
51.             e.printStackTrace();
52.
53.         } catch(ClassNotFoundException e) {
54.
55.             e.printStackTrace();
56.
57.         }
58.
59.     }
60.
61. }
```

Na Figura 5 também é mostrada a utilização dos métodos *start*() e *stop*() da classe *StopWatch*, cujo código é o definido na Figura 6.

**Figura 6 – Classe StopWatch**

```java
public class StopWatch {

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


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


    public void stop() {
        this.stopTime = System.currentTimeMillis();
        this.running = false;
    }


    //elaspsed time in milliseconds
    public long getElapsedTime() {
        long elapsed;
        if (running) {
             elapsed = (System.currentTimeMillis() - startTime);
        }
        else {
            elapsed = (stopTime - startTime);
        }
        return elapsed;
    }


    //elaspsed time in seconds
    public long getElapsedTimeSecs() {
        long elapsed;
        if (running) {
            elapsed = ((System.currentTimeMillis() - startTime) / 1000);
        }
        else {
            elapsed = ((stopTime - startTime) / 1000);
        }
        return elapsed;
    }




    //sample usage
    public static void main(String[] args) {
        StopWatch s = new StopWatch();
        s.start();
        //code you want to time goes here
        s.stop();
        System.out.println("elapsed time in milliseconds: " + s.getElapsedTime());
    }
}
```

Cada operação solicitada foi repetida cinco vezes com o mesmo volume de dados, tendo sido adotado e anotado como resultado do tempo despendido a média aritmética apurada entre as cinco repetições.

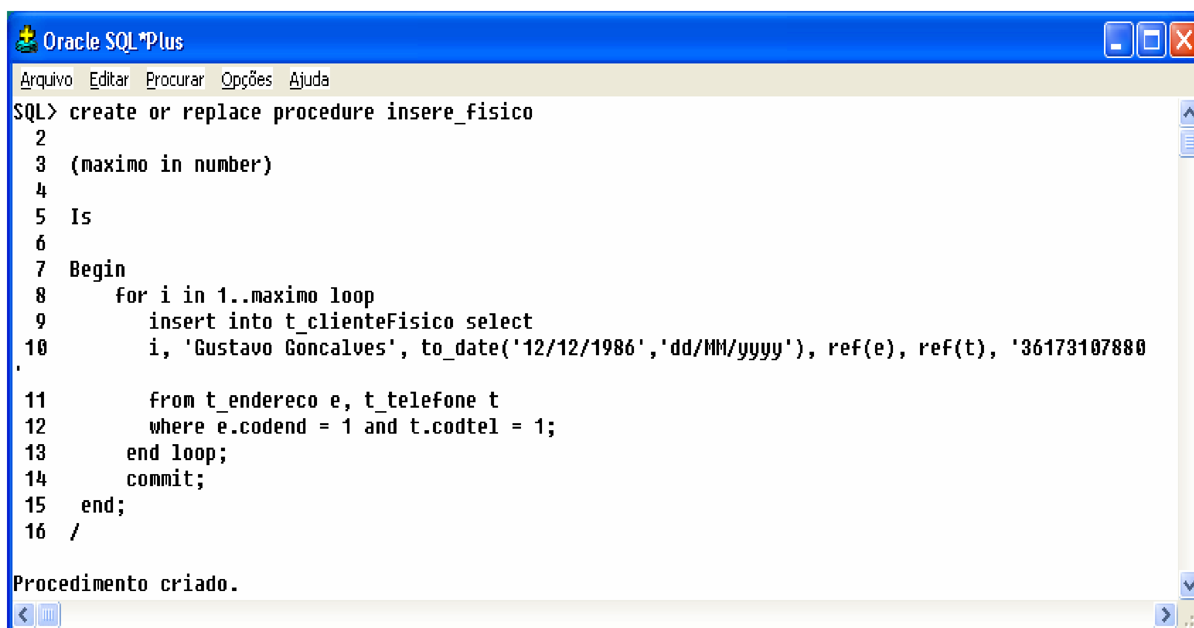
Na etapa de inserção, a diferenciação dos dados foi determinada apenas pelo código, sendo este atribuído de acordo com o índice do laço. Os demais atributos foram repetidos, como mostrado na Figura 7.

**Figura 7 – Exemplo de inserção de dados**

```
Oracle SQL*Plus
Arquivo  Editar  Procurar  Opções  Ajuda
SQL> select * from t_clientefisico;

CODCLIENTE      NOME                                                DATANASC
--------------- --------------------------------------------------- --------
ENDERECOCLIENTE
----------------------------------------------------------------------------------
TELEFONECLI
----------------------------------------------------------------------------------
CPF
---------------
1               Gustavo Goncalves                                   12/12/86
0000220208DABC2923832F4E7F91C128F56DFD5DCE228C51D58F724D86AC9868BE0696034C
0000220208925C4BB22E7A4BD991EF7E2CFC1F14F17BA891A5BFEE4791894B803154C0F5A1
36173107880


CODCLIENTE      NOME                                                DATANASC
--------------- --------------------------------------------------- --------
ENDERECOCLIENTE
----------------------------------------------------------------------------------
TELEFONECLI
----------------------------------------------------------------------------------
CPF
---------------
2               Gustavo Goncalves                                   12/12/86
0000220208DABC2923832F4E7F91C128F56DFD5DCE228C51D58F724D86AC9868BE0696034C
0000220208925C4BB22E7A4BD991EF7E2CFC1F14F17BA891A5BFEE4791894B803154C0F5A1
36173107880


CODCLIENTE      NOME                                                DATANASC
--------------- --------------------------------------------------- --------
ENDERECOCLIENTE
----------------------------------------------------------------------------------
TELEFONECLI
----------------------------------------------------------------------------------
CPF
---------------
3               Gustavo Goncalves                                   12/12/86
0000220208DABC2923832F4E7F91C128F56DFD5DCE228C51D58F724D86AC9868BE0696034C
0000220208925C4BB22E7A4BD991EF7E2CFC1F14F17BA891A5BFEE4791894B803154C0F5A1
36173107880
```

Na Versão 1, as inserções foram realizadas por meio da *stored procedure*, envolvendo um laço que, além de ser responsável pelas repetições do comando de inserção, teve a função de atribuir um código diferente (incremental) a cada uma delas, como mostra o trecho de código apresentado na Figura 8.

**Figura 8 – Procedimento de inserção com stored procedure (Versão 1)**

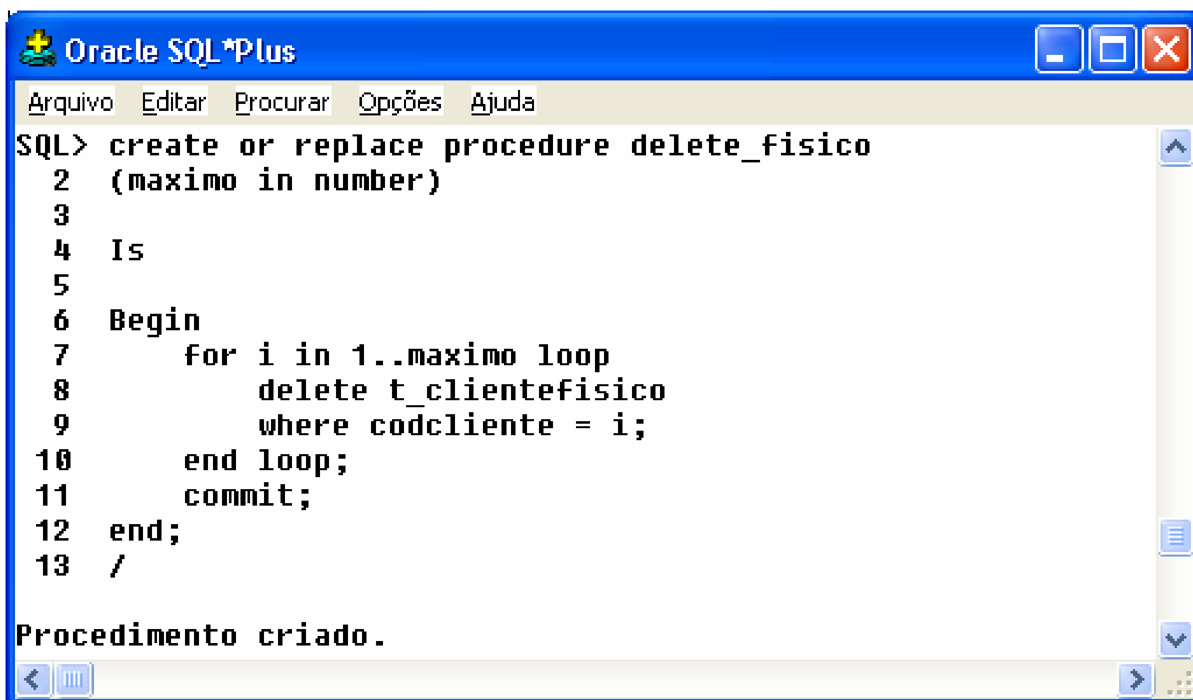



Na Versão 2, as inserções foram realizadas por meio de procedimento implementado em JAVA, também envolvendo um laço que, além de ser responsável pelas repetições do comando de inserção, teve a função de atribuir um código diferente (incremental) a cada uma delas, como mostra o trecho de código já apresentado na Figura 5.

Na etapa de remoção, o volume de dados removidos foi correspondente ao de dados inseridos.

Na Versão 1, as remoções foram realizadas por meio da *stored procedure*, envolvendo um laço responsável pela remoção de tantos Clientes quantos tivessem o código (codcliente) compreendido no intervalo do laço passado como parâmetro, como demonstra a Figura 9.

**Figura 9 – Procedimento de remoção com stored procedure (Versão 1)**



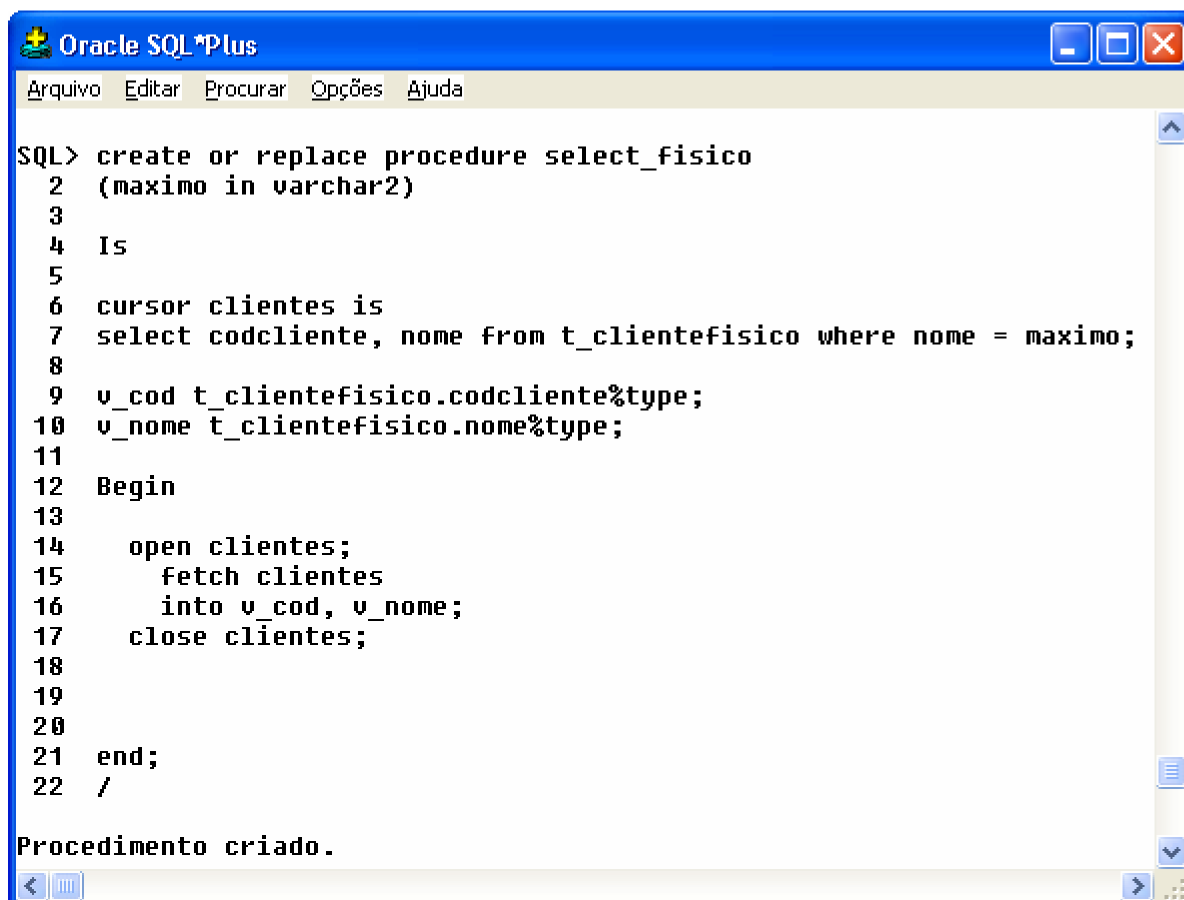

Na Versão 2, as remoções foram realizadas por meio de procedimento implementado em JAVA, também envolvendo um laço responsável pela remoção de tantos Clientes quantos tivessem o código (*codcliente*) compreendido no intervalo do laço passado como parâmetro, como demonstra a Figura 10.

**Figura 10 – Procedimento de remoção implementado em Java (Versão 2)**

```
4.  public class Delete2 {
5.
6.
7.      public static void main(String... args){
8.      StopWatch s = new StopWatch();// cria objeto da classe cronômetro
9.      try{
10.
11.
12.     //conexão co banco de dados
13.     Class.forName("oracle.jdbc.driver.OracleDriver");
14.     Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:vica",
15.             "scott", "tiger");
16.     System.out.println("deu certo");
17.
18.
19.
20.
21.     //declaração do prepared statement
22.     PreparedStatement pre2 = conn.prepareStatement("delete from " +
23.             "t_clientefisico where codcliente = ?");
24.
25.     s.start();//iniciacontagem do tempo
26.
27.     for (int i=1; i<=10; i++ ) {//looping com 10 iterações
28.
29.     pre2.setString(1, Integer.toString(i)); //seta o valor variavel do prepared statemente
30.                                             //com o indice do looping a cada iteração
31.
32.     pre2.executeUpdate();
33.
34.       }
35.
36.
37.             s.stop();// fecha a contagem do tempo
38.
39.
40.             System.out.println("procedure executado");
41.             JOptionPane.showMessageDialog(null,"Tempo de execução"+ s.getElapsedTime());
42.
43. conn.close();//fecha a conexão com o banco
44.         } catch(SQLException e) {
45.
46.             e.printStackTrace();
47.
48.         } catch(ClassNotFoundException e) {
49.
50.             e.printStackTrace();
51.
52.         }
53.
54.     }
55.
56. }
```

Na etapa de recuperação, houve solicitação de seleção de dados de acordo com o nome do Cliente (nome), segundo parâmetro fornecido a um procedimento criado com a finalidade de retornar os respectivos dados. Os trechos de código referentes à recuperação de dados na Versão 1 e na Versão 2 encontram-se especificados nas Figuras 11 e 12.

**Figura 11 – Procedimento de recuperação com stored procedure (Versão 1)**



```
SQL> create or replace procedure select_fisico
  2  (maximo in varchar2)
  3
  4  Is
  5
  6  cursor clientes is
  7  select codcliente, nome from t_clientefisico where nome = maximo;
  8
  9  v_cod t_clientefisico.codcliente%type;
 10  v_nome t_clientefisico.nome%type;
 11
 12  Begin
 13
 14    open clientes;
 15      fetch clientes
 16      into v_cod, v_nome;
 17    close clientes;
 18
 19
 20
 21  end;
 22  /

Procedimento criado.
```

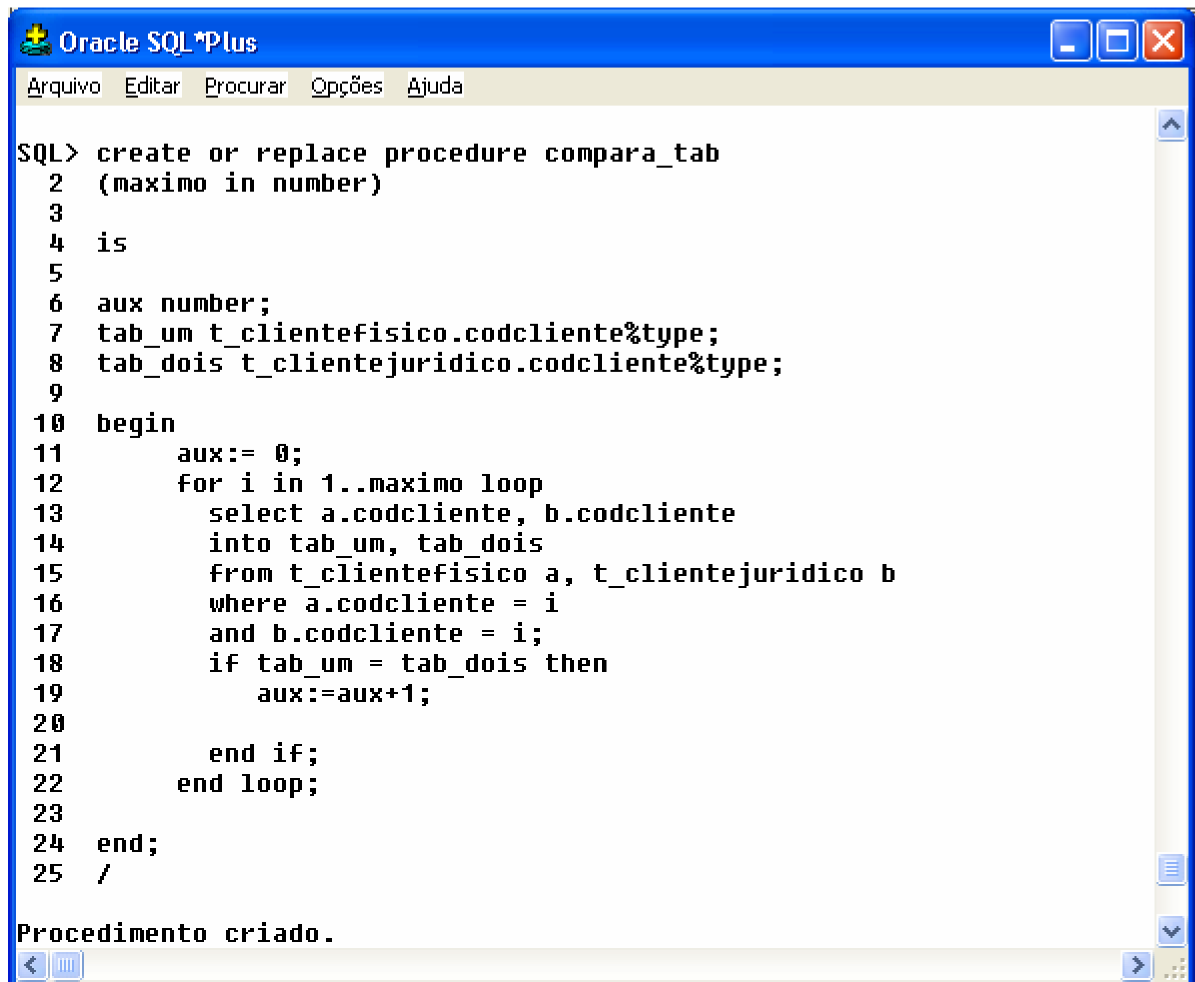
**Figura 12 – Procedimento de recuperação implementado em Java (Versão 2)**

```
1.  public class Recuperacao2 {
2.
3.
4.      public static void main(String... args){
5.          StopWatch s = new StopWatch();
6.          try{
7.
8.
9.
10. Class.forName("oracle.jdbc.driver.OracleDriver");
11. Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:vica"
12.                     , "scott", "tiger");
13. System.out.println("deu certo");
14.
15.
16.
17.
18.
19. PreparedStatement pre2 = conn.prepareStatement("Select * from t_clientefisico" +
20. " where nome=?");
21.
22.
23.
24.
25.
26.   pre2.setString(1,"Gustavo Goncalves"); //seta o valor variavel do prepared
27.                                          //de modo que todas as linhas sejam
28.                                         //selecionadas
29.   ResultSet rSet = null;
30.   s.start();//iniciacontagem do tempo
31.             pre2.executeUpdate();
32.             rSet = pre2.executeQuery();
33.             while(rSet.next()){
34.
35.                 int a_codcliente = rSet.getInt(1);
36.                 String a_nome = rSet.getString(2);
37.
38.
39.             }
40.
41.             s.stop();// fecha a contagem do tempo
42.
43.
44. System.out.println("procedure executado");
45. JOptionPane.showMessageDialog(null,"Tempo de execução"+ s.getElapsedTime());
46.
47. conn.close();
48.         } catch(SQLException e) {
49.
50.             e.printStackTrace();
51.
52.         } catch(ClassNotFoundException e) {
53.
54.             e.printStackTrace();
55.
56.         }
57.
58.     }
59.
60. }
```

Na etapa de comparação, foi solicitada a verificação de ocorrência de códigos idênticos nas Tabelas t_clientefisico e t_clientejuridico. Para tanto, foi criado um

procedimento, que também teve por finalidade armazenar a soma das ocorrências verificadas, conforme trechos de código contidos nas Figuras 13 e 14.

**Figura 13 – Procedimento de comparação com stored procedure (Versão 1)**



```
Oracle SQL*Plus
Arquivo Editar Procurar Opções Ajuda

SQL> create or replace procedure compara_tab
  2  (maximo in number)
  3
  4  is
  5
  6  aux number;
  7  tab_um t_clientefisico.codcliente%type;
  8  tab_dois t_clientejuridico.codcliente%type;
  9
 10  begin
 11       aux:= 0;
 12       for i in 1..maximo loop
 13         select a.codcliente, b.codcliente
 14         into tab_um, tab_dois
 15         from t_clientefisico a, t_clientejuridico b
 16         where a.codcliente = i
 17         and b.codcliente = i;
 18         if tab_um = tab_dois then
 19            aux:=aux+1;
 20
 21         end if;
 22       end loop;
 23
 24  end;
 25  /

Procedimento criado.
```

**Figura 14 – Procedimento de comparação implementado em Java (Versão 2)**

```
1.  public class Comparacao2 {
2.
3.
4.      public static void main(String... args){
5.          StopWatch s = new StopWatch();
6.          try{
7.
8.
9.
10. Class.forName("oracle.jdbc.driver.OracleDriver");
11. Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:vica",
12.         "scott", "tiger");
13. System.out.println("deu certo");
14.
15.
16.
17.
18.
19.
20.             int aux = 0;// variavel que armazena quantidade de ocorrências
21.
22.
23.
24.
25.             //prepared statement que compara os códigos de duas tabelas
26.             PreparedStatement pre2 = conn.prepareStatement("select a.codcliente, " +
27.                     "b.codcliente from t_clientefisico a, " +
28.                     "t_clientejuridico b where a.codcliente = " +
29.                     "? and b.codcliente = ?");
30.
31.              ResultSet rSet = null;
32.               s.start();
33.             for (int i=1; i<=10000; i++ ) {// laço com 10000 comparações
34.
35.              pre2.setString(1, Integer.toString(i));
36.              pre2.setString(2, Integer.toString(i));
37.
38.               pre2.executeUpdate();
39.               rSet = pre2.executeQuery();                 //método de comparação
40.                                                           //com recursos JAVA
41.               while(rSet.next()){
42.
43.                   int a_codcliente = rSet.getInt(1);
44.                   int b_codcliente = rSet.getInt(2);
45.                   if(a_codcliente == b_codcliente){
46.
47.                       aux = aux +1;
48.                   }
49.               }
50.             }
51.
52.               s.stop();
53.               System.out.println("procedure executado");
54.               JOptionPane.showMessageDialog(null,"Tempo de execução"+ s.getElapsedTime());
55.
56. conn.close();
57.         } catch(SQLException e) {
58.
59.               e.printStackTrace();
60.
61.         } catch(ClassNotFoundException e) {
62.
63.               e.printStackTrace();
64.         }
65.     }
66. }
```

Na etapa de atualização, foi solicitada a alteração de nomes (nome) na Tabela t_clientefisico. Para tanto, foi criado um procedimento com a finalidade de alterar o nome (nome) “Gustavo Gonçalves” – que foi atributo inserido em todas as linhas da Tabela t_clientefisico (FIGURA 7) – para “Joaquim”. Os trechos de código correspondentes são os mostrados nas Figuras 15 e 16.

**Figura 15 – Procedimento de atualização com stored procedure (Versão 1)**

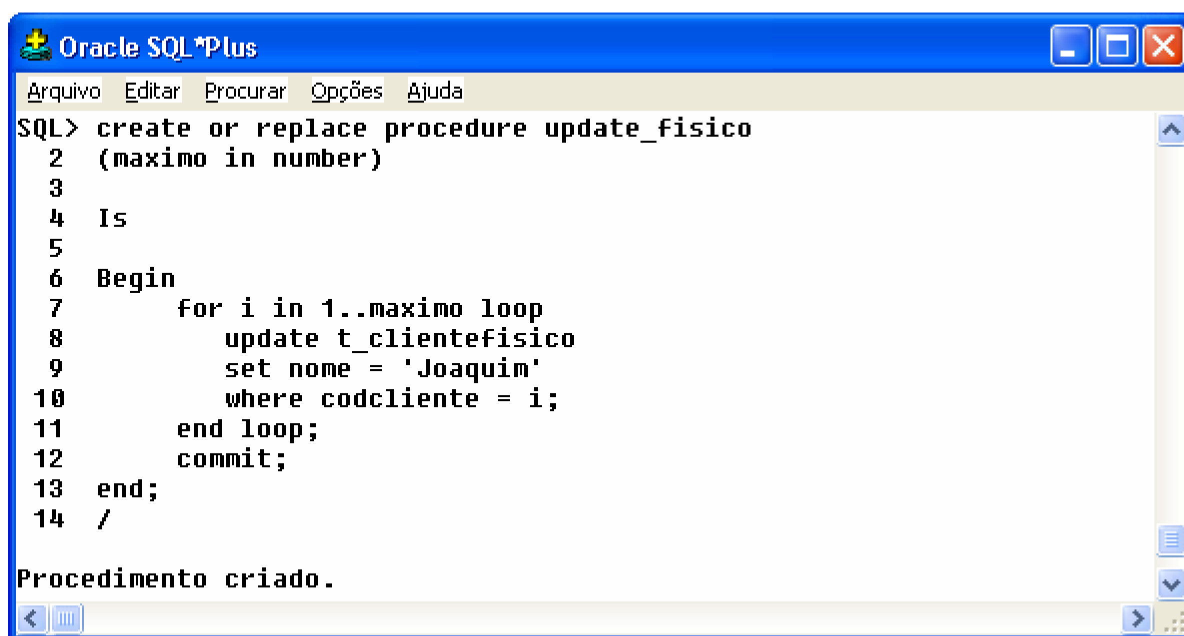


**Figura 16 – Procedimento de atualização implementado em Java (Versão 2)**

```
1.  public class Update2 {
2.
3.
4.      public static void main(String... args){
5.          StopWatch s = new StopWatch();
6.          try{
7.
8.
9.
10. Class.forName("oracle.jdbc.driver.OracleDriver");
11. Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:vica",
12.           "scott", "tiger");
13. System.out.println("deu certo");
14.
15.
16.
17.
18.
19. PreparedStatement pre2 = conn.prepareStatement("update t_clientefisico set nome" +
20.           " = 'Joaquim' where codcliente = ?");
21.
22. s.start();//iniciacontagem do tempo
23.
24. for (int i=1; i<=1000; i++ ) {//exemplo de looping com 1000 iterações
25.
26. pre2.setString(1, Integer.toString(i)); //seta o valor variavel do prepared
27.                                         //statemente com o indice d looping a cada iteração
28.
29. pre2.executeUpdate();
30.
31.           }
32.
33.
34.             s.stop();// fecha a contagem do tempo
35.
36.
37.    System.out.println("procedure executado");
38.    JOptionPane.showMessageDialog(null,"Tempo de execução"+ s.getElapsedTime());
39.
40. conn.close();
41.         } catch(SQLException e) {
42.
43.             e.printStackTrace();
44.
45.         } catch(ClassNotFoundException e) {
46.
47.             e.printStackTrace();
48.
49.         }
50.
51.     }
52.
53. }
```

# 4 RESULTADOS OBTIDOS

Executadas as avaliações, foram obtidos os resultados que são declarados nas Seções 4.1 e 4.2, comparados na Seção 4.3 e finalmente analisados na Seção 4.4.

## 4.1 Resultados Obtidos na Avaliação da Versão 1 da Aplicação de BDOO

Na avaliação relativa ao procedimento de inserção de 10, 100, 1.000 e 10.000 dados, a Versão 1 cumpriu a tarefa nos tempos médios de 9,2, 15,2, 175 e 1631,4 milissegundos, respectivamente. Na Tabela 1 são relacionados os tempos cronometrados em cada uma das cinco repetições do teste e suas correspondentes médias.

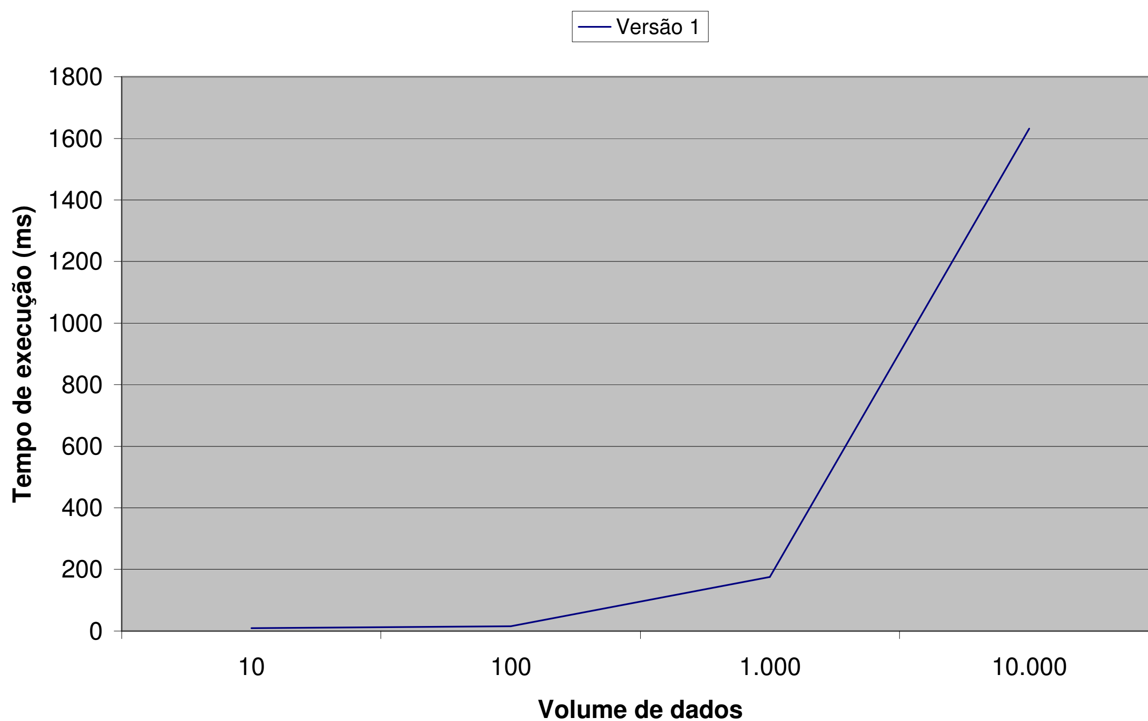
**Tabela 1 – Resultados obtidos no procedimento de inserção (Versão 1)**

| Volume de Dados | Tempo 1 | Tempo 2 | Tempo 3 | Tempo 4 | Tempo 5 | Tempo Médio |
|---|---|---|---|---|---|---|
| **10** | 16 | 0 | 0 | 15 | 15 | 9,2 |
| **100** | 16 | 15 | 15 | 15 | 15 | 15,2 |
| **1.000** | 203 | 172 | 172 | 156 | 172 | 175 |
| **10.000** | 1641 | 1594 | 1578 | 1672 | 1672 | 1631,4 |

tempo = em milissegundos

No Gráfico 1 é mostrado o comportamento do procedimento de inserção de acordo com  a quantidade de dados inseridos na Versão aqui tratada.

Gráfico 1 - Comportamento do procedimento de inserção (Versão 1)




Na avaliação relativa ao procedimento de remoção de 10, 100, 1.000 e 10.000 dados, a Versão 1 cumpriu a tarefa nos tempos médios de 12,4, 16, 337,6 e 28340,2 milissegundos, respectivamente. Na Tabela 2 são relacionados os tempos cronometrados em cada uma das cinco repetições do teste e suas correspondentes médias.

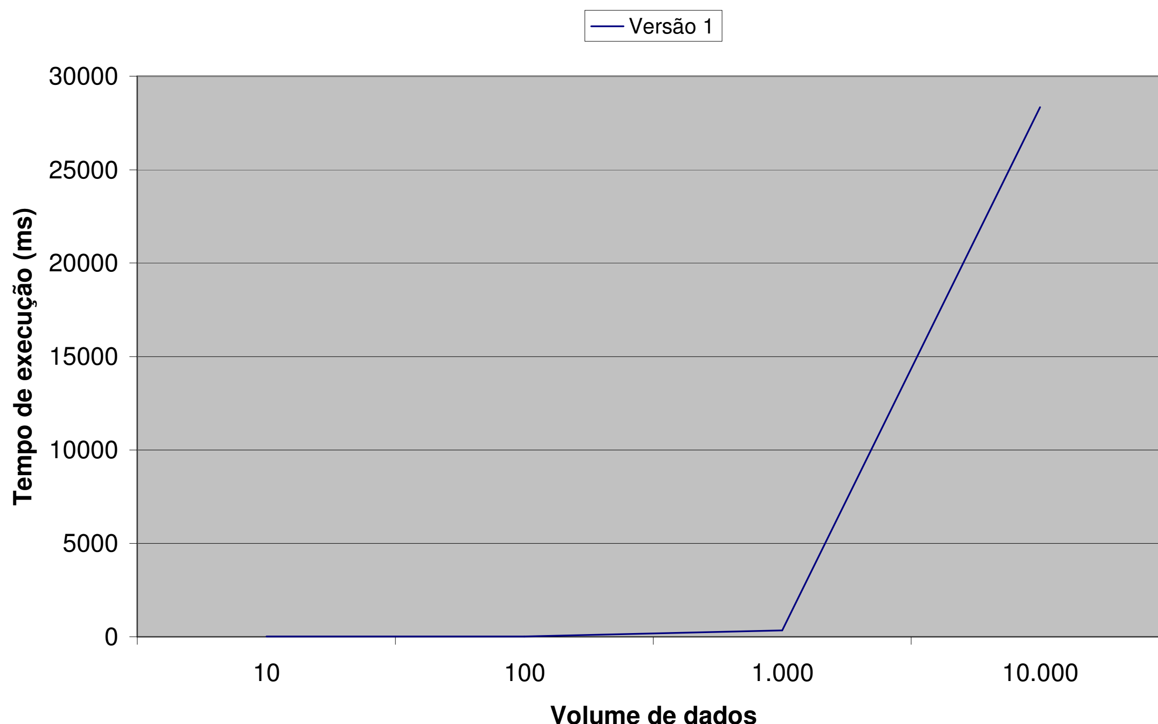
**Tabela 2 – Resultados obtidos no procedimento de remoção (Versão 1)**

| Volume de Dados | Tempo 1 | Tempo 2 | Tempo 3 | Tempo 4 | Tempo 5 | Média |
|---|---|---|---|---|---|---|
| **10** | 16 | 0 | 15 | 15 | 16 | 12,4 |
| **100** | 16 | 16 | 16 | 16 | 16 | 16 |
| **1.000** | 375 | 328 | 328 | 329 | 328 | 337,6 |
| **10.000** | 27796 | 28516 | 28546 | 28437 | 28406 | 28340,2 |

tempo = em milissegundos

No Gráfico 2 é mostrado o comportamento do procedimento de remoção de acordo com a quantidade de dados inseridos na Versão aqui tratada.

Gráfico 2 - Comportamento do procedimento de remoção (Versão 1)




Na avaliação relativa ao procedimento de recuperação de 10, 100, 1.000 e 10.000 dados, a Versão 1 cumpriu a tarefa nos tempos médios de 0, 0, 6,4 e 9,6 milissegundos, respectivamente. Na Tabela 3 são relacionados os tempos cronometrados em cada uma das cinco repetições do teste e suas correspondentes médias.

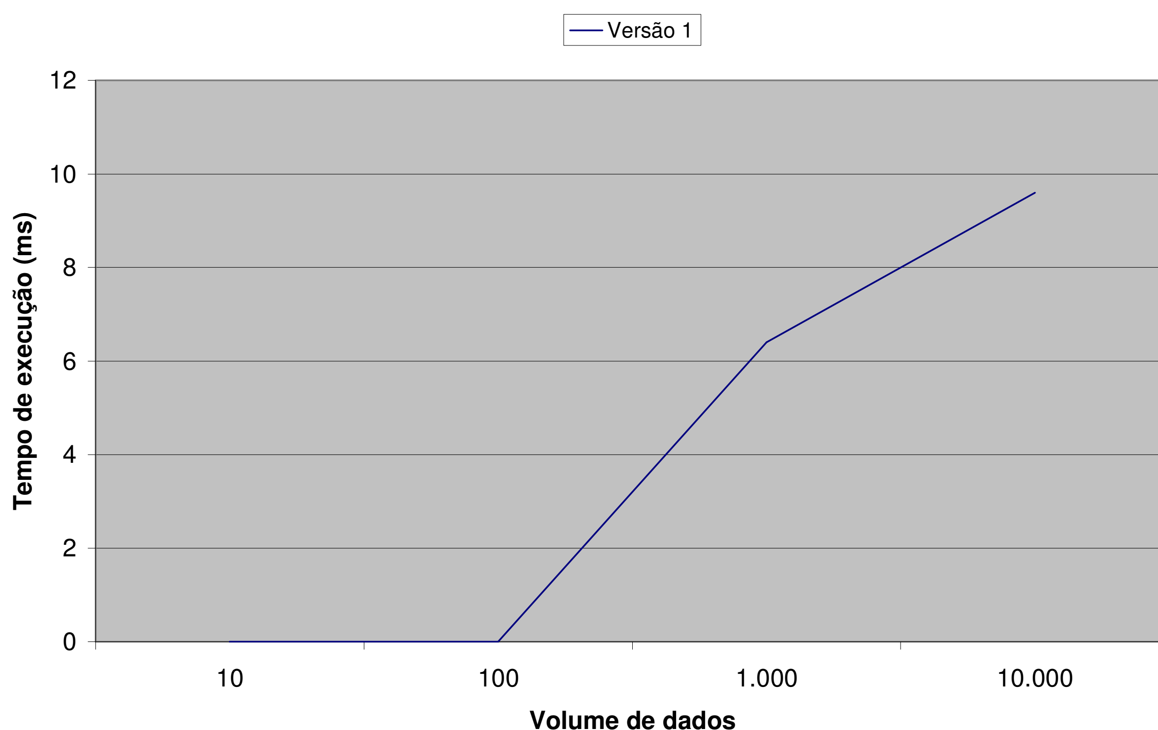
**Tabela 3 – Resultados obtidos no procedimento de recuperação (Versão 1)**

| Volume de Dados | Tempo 1 | Tempo 2 | Tempo 3 | Tempo 4 | Tempo 5 | Média |
|---|---|---|---|---|---|---|
| **10** | 0 | 0 | 0 | 0 | 0 | 0 |
| **100** | 0 | 0 | 0 | 0 | 0 | 0 |
| **1.000** | 16 | 16 | 0 | 0 | 0 | 6,4 |
| **10.000** | 16 | 0 | 0 | 16 | 16 | 9,6 |

tempo = em milissegundos

Com base nestes resultados, foi gerado o Gráfico 3 que mostra o comportamento do procedimento de recuperação de acordo com a quantidade de dados inseridos na Versão aqui tratada.

Gráfico 3 - Comportamento do procedimento de recuperação (Versão 1)




Na avaliação relativa ao procedimento de alteração de 10, 100, 1.000 e 10.000 dados, a Versão 1 cumpriu a tarefa nos tempos médios de 3, 15,2, 468,6 e 45512,6 milissegundos, respectivamente. Na Tabela 4 são relacionados os tempos cronometrados em cada uma das cinco repetições do teste e suas correspondentes médias.

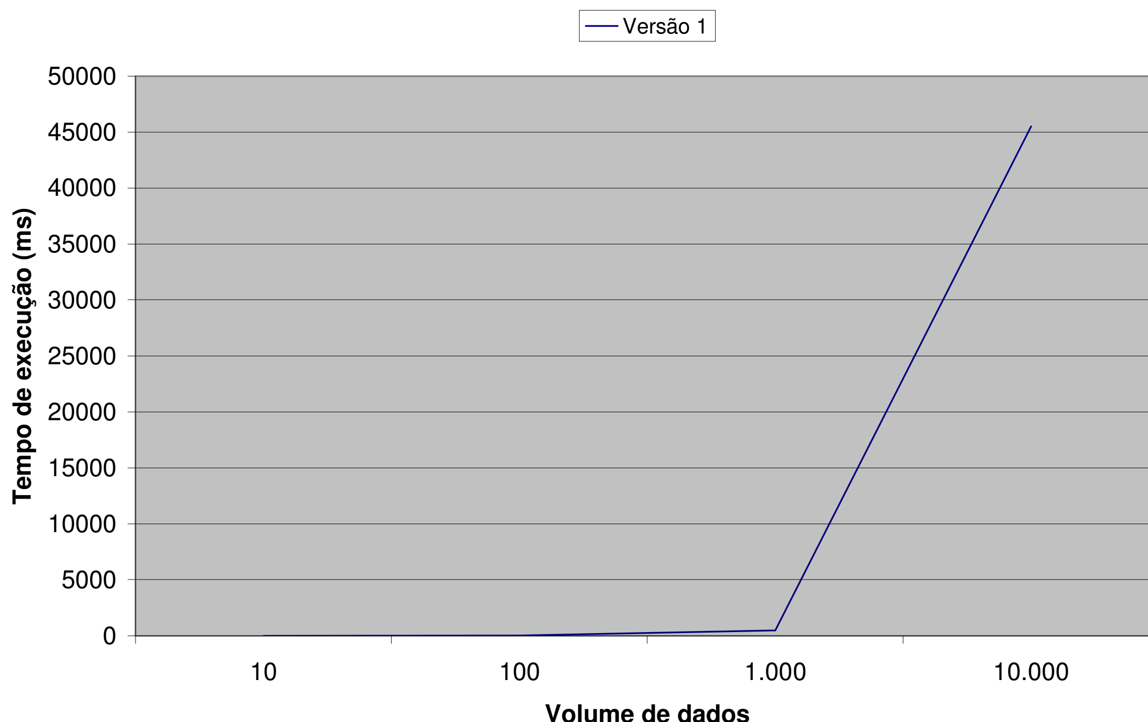
**Tabela 4 – Resultados obtidos no procedimento de alteração (Versão 1)**

| Volume de Dados | Tempo 1 | Tempo 2 | Tempo 3 | Tempo 4 | Tempo 5 | Média |
|---|---|---|---|---|---|---|
| **10** | 0 | 0 | 15 | 0 | 0 | 3 |
| **100** | 16 | 15 | 15 | 15 | 15 | 15,2 |
| **1.000** | 453 | 484 | 484 | 469 | 453 | 468,6 |
| **10.000** | 43547 | 46266 | 45797 | 46109 | 45844 | 45512,6 |

tempo = em milissegundos

Com estes resultados, o comportamento do procedimento de alteração, de acordo com a quantidade de dados inseridos na Versão aqui tratada, é o representado no Gráfico 4.

Gráfico 4 - Comportamento do procedimento de alteração (Versão 1)




Na avaliação relativa ao procedimento de comparação de 10, 100, 1.000 e 10.000 dados, a Versão 1 cumpriu a tarefa nos tempos médios de 0, 15,4, 774,6 e 79656,4 milissegundos, respectivamente. Na Tabela 5 são relacionados os tempos cronometrados em cada uma das cinco repetições do teste e suas correspondentes médias.

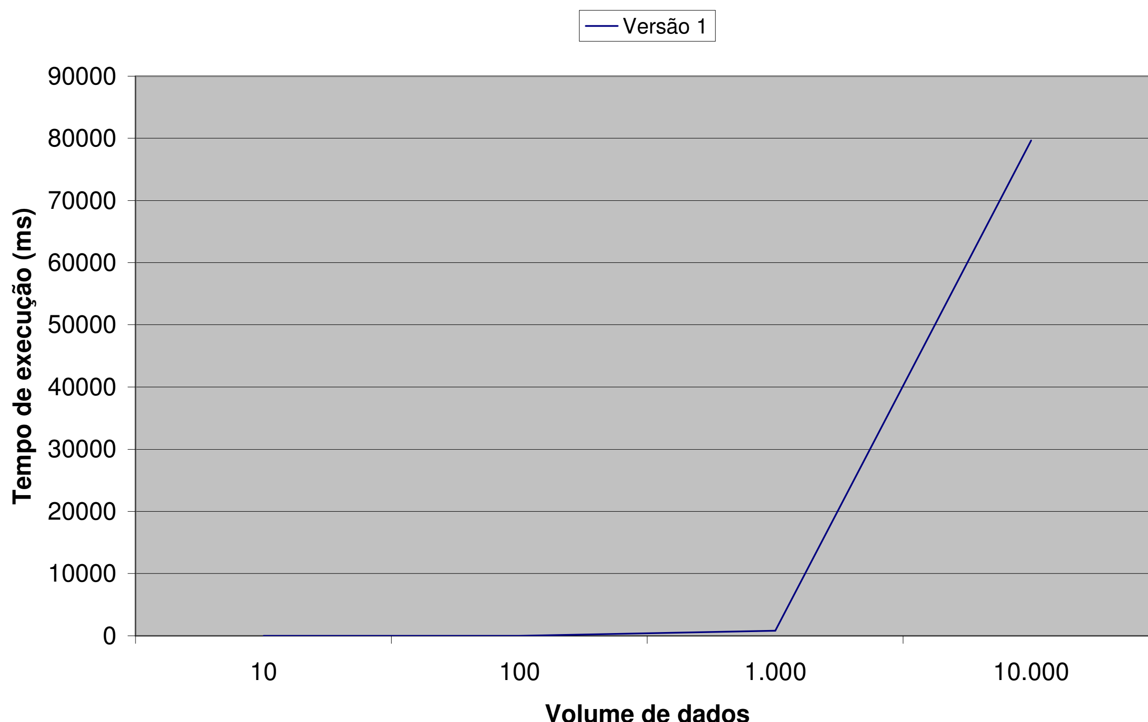
**Tabela 5 – Resultados obtidos no procedimento de comparação (Versão 1)**

| Volume de Dados | Tempo 1 | Tempo 2 | Tempo 3 | Tempo 4 | Tempo 5 | Média |
|---|---|---|---|---|---|---|
| **10** | 0 | 0 | 0 | 0 | 0 | 0 |
| **100** | 15 | 16 | 16 | 15 | 15 | 15,4 |
| **1.000** | 765 | 796 | 765 | 765 | 782 | 774,6 |
| **10.000** | 79844 | 79797 | 79656 | 79266 | 79719 | 79656,4 |

tempo = em milissegundos

No Gráfico 5 é mostrado o comportamento do procedimento de comparação de acordo com a quantidade de dados inseridos na Versão aqui tratada.

Gráfico 5 - Comportamento do procedimento de comparação (Versão 1)




## 4.2 Resultados Obtidos na Avaliação da Versão 2 da Aplicação de BDOO

Na avaliação relativa ao procedimento de inserção de 10, 100, 1.000 e 10.000 dados, a Versão 2 executou a tarefa nos tempos médios de 12,6, 63, 562,8 e 5094 milissegundos, respectivamente. Na Tabela 6 são relacionados os tempos cronometrados em cada uma das cinco repetições do teste e suas correspondentes médias.

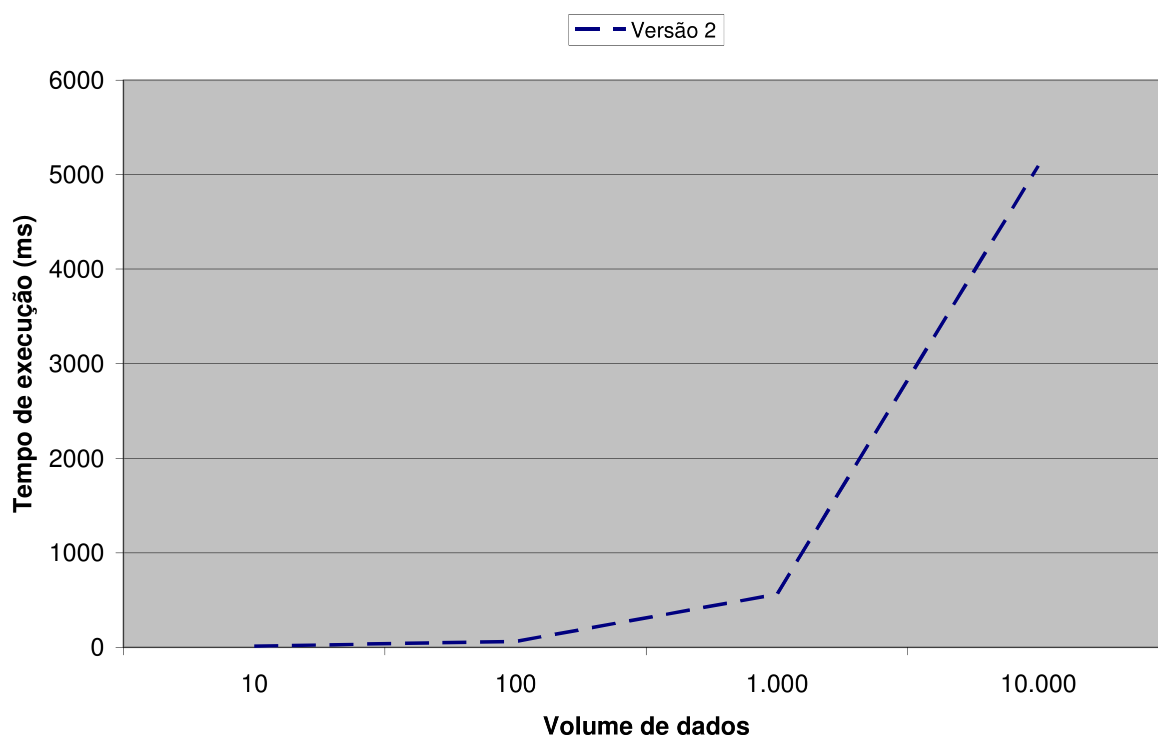
**Tabela 6 – Resultados obtidos no procedimento de inserção (Versão 2)**

| Volume de Dados | Tempo 1 | Tempo 2 | Tempo 3 | Tempo 4 | Tempo 5 | Média |
|---|---|---|---|---|---|---|
| **10** | 16 | 15 | 0 | 16 | 16 | 12,6 |
| **100** | 63 | 63 | 63 | 63 | 63 | 63 |
| **1.000** | 547 | 547 | 610 | 563 | 547 | 562,8 |
| **10.000** | 5922 | 4750 | 4829 | 5156 | 4813 | 5094 |

tempo = em milissegundos

No Gráfico 6 é mostrado o comportamento do procedimento de inserção de acordo com a quantidade de dados inseridos na Versão aqui tratada.

**Gráfico 6 - Comportamento do procedimento de inserção (Versão 2)**




Na avaliação relativa ao procedimento de remoção de 10, 100, 1.000 e 10.000 dados, a Versão 2 executou a tarefa nos tempos médios de 15,6, 62,6, 631,6 e 22153,2 milissegundos, respectivamente. Na Tabela 7 são relacionados os tempos cronometrados em cada uma das cinco repetições do teste e suas correspondentes médias.

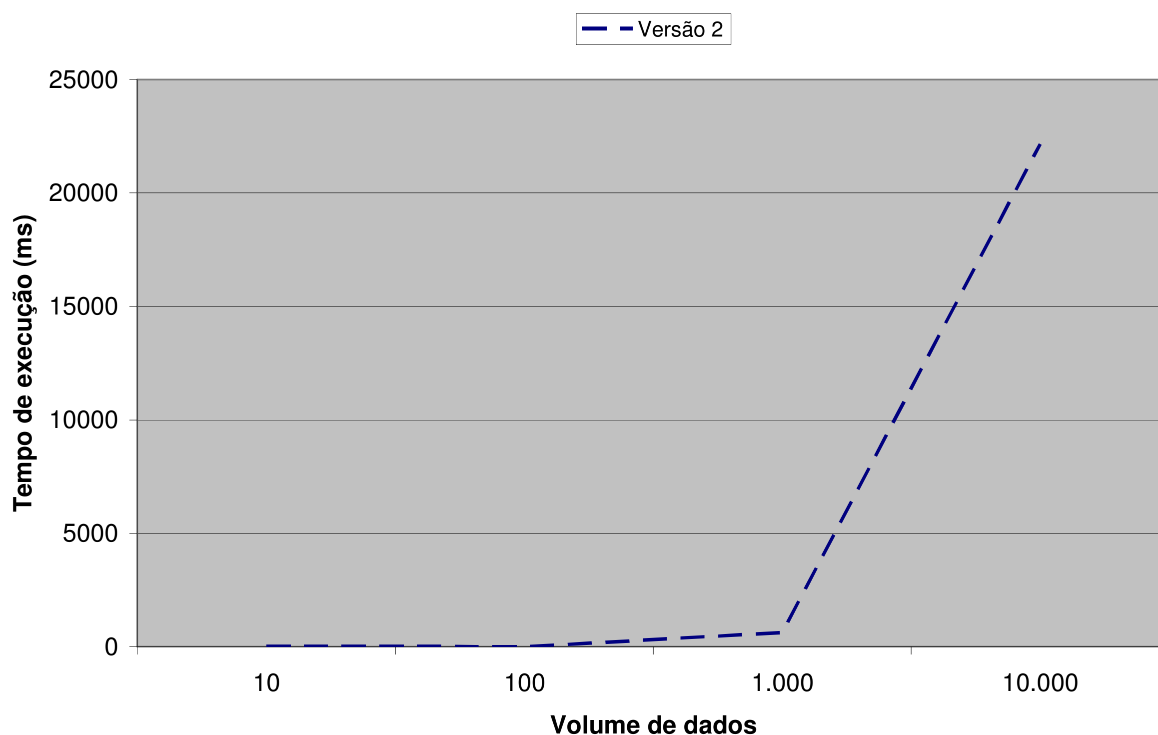
**Tabela 7 – Resultados obtidos no procedimento de remoção (Versão 2)**

| Volume de Dados | Tempo 1 | Tempo 2 | Tempo 3 | Tempo 4 | Tempo 5 | Média |
|---|---|---|---|---|---|---|
| **10** | 16 | 15 | 31 | 0 | 16 | 15,6 |
| **100** | 46 | 63 | 63 | 63 | 78 | 62,6 |
| **1.000** | 625 | 610 | 641 | 672 | 610 | 631,6 |
| **10.000** | 20500 | 22641 | 23516 | 22078 | 22031 | 22153,2 |

tempo = em milissegundos

No Gráfico 7 é mostrado o comportamento do procedimento de remoção de acordo com a quantidade de dados inseridos na Versão aqui tratada.

**Gráfico 7 - Comportamento do procedimento de remoção (Versão 2)**




Na avaliação relativa ao procedimento de recuperação de 10, 100, 1.000 e 10.000 dados, a Versão 2 executou a tarefa nos tempos médios de 6,8, 15,4, 56 e 262,4 milissegundos, respectivamente. Na Tabela 8 são relacionados os tempos cronometrados em cada uma das cinco repetições do teste e suas correspondentes médias.

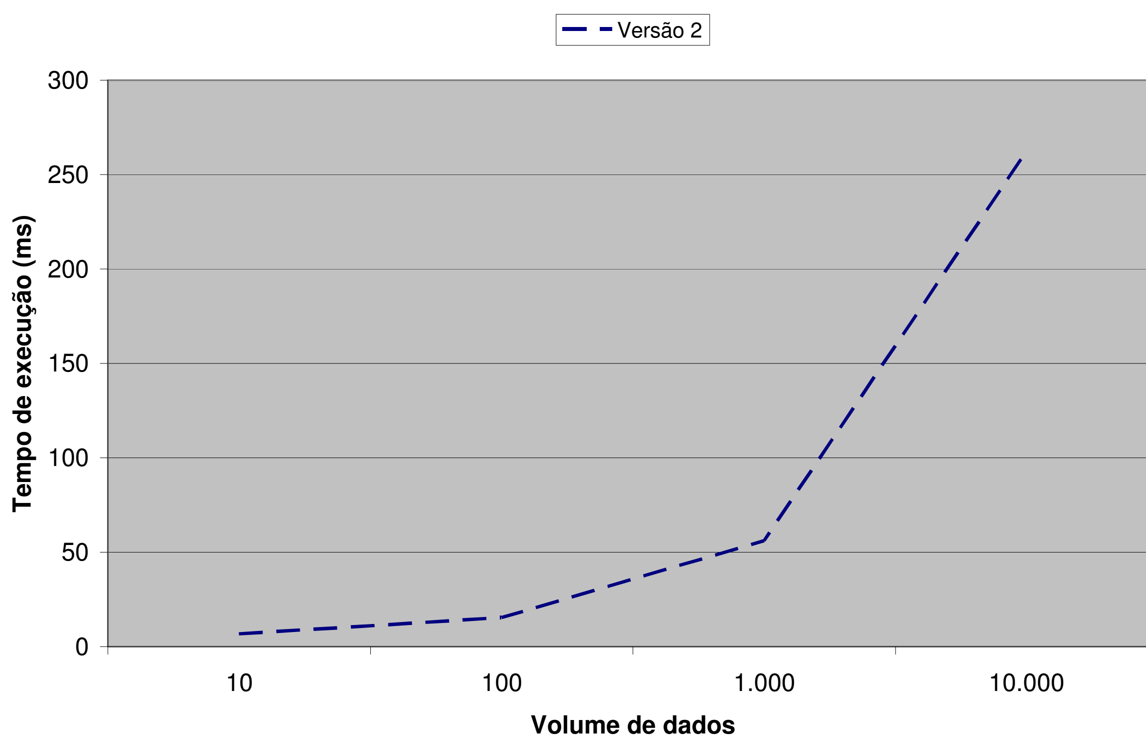
**Tabela 8 – Resultados obtidos no procedimento de recuperação (Versão 2)**

| Volume de Dados | Tempo 1 | Tempo 2 | Tempo 3 | Tempo 4 | Tempo 5 | Média |
|---|---|---|---|---|---|---|
| **10** | 0 | 0 | 16 | 0 | 18 | 6,8 |
| **100** | 15 | 16 | 16 | 15 | 15 | 15,4 |
| **1.000** | 47 | 62 | 46 | 63 | 62 | 56 |
| **10.000** | 265 | 250 | 266 | 265 | 266 | 262,4 |

tempo = em milissegundos

No Gráfico 8 é mostrado o comportamento do procedimento de recuperação de acordo com a quantidade de dados inseridos na Versão aqui tratada.

**Gráfico 8 - Comportamento do procedimento de recuperação (Versão 2)**




Na avaliação relativa ao procedimento de alteração de 10, 100, 1.000 e 10.000 dados, a Versão 2 executou a tarefa nos tempos médios de 9,2, 53,2, 653,2 e 29409,4 milissegundos, respectivamente. Na Tabela 9 são relacionados os tempos cronometrados em cada uma das cinco repetições do teste e suas correspondentes médias.

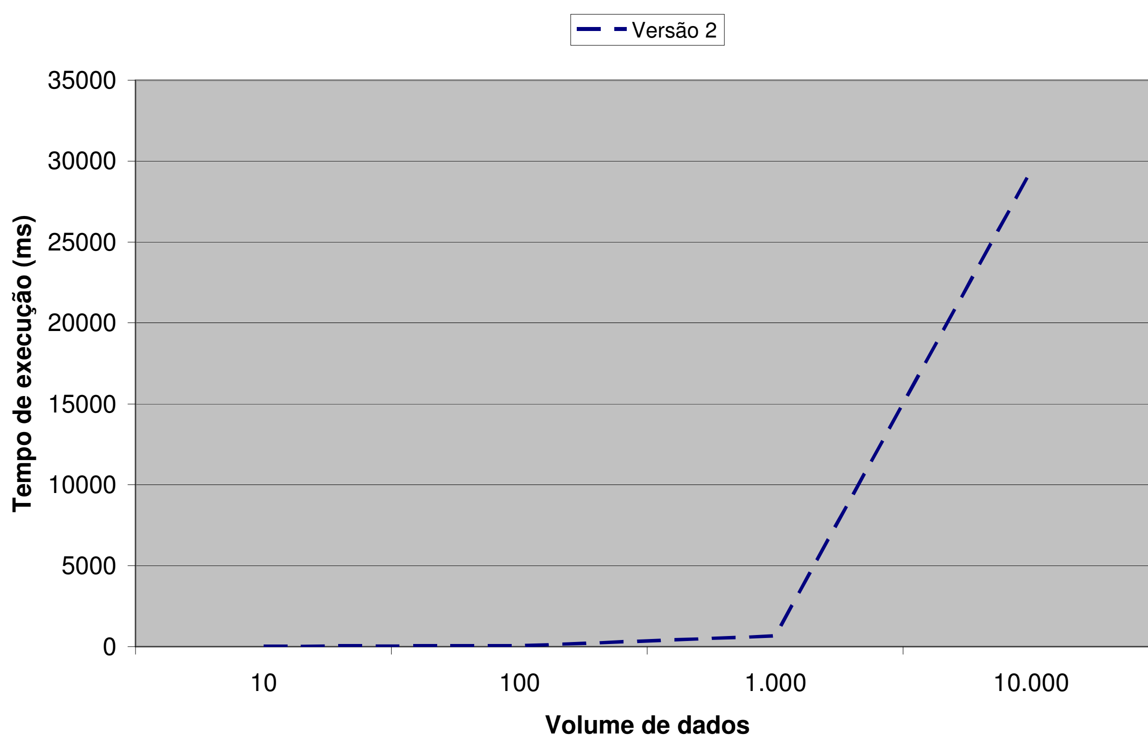
**Tabela 9 – Resultados obtidos no procedimento de alteração (Versão 2)**

| Volume de Dados | Tempo 1 | Tempo 2 | Tempo 3 | Tempo 4 | Tempo 5 | Média |
|---|---|---|---|---|---|---|
| **10** | 0 | 16 | 0 | 15 | 15 | 9,2 |
| **100** | 47 | 47 | 63 | 62 | 47 | 53,2 |
| **1.000** | 610 | 672 | 656 | 672 | 656 | 653,2 |
| **10.000** | 27171 | 30579 | 29266 | 30297 | 2297334 | 29409,4 |

tempo = em milissegundos

No Gráfico 9 é mostrado o comportamento do procedimento de alteração de acordo com a quantidade de dados inseridos na Versão aqui tratada.

**Gráfico 9 - Comportamento do procedimento de alteração (Versão 2)**




Na avaliação relativa ao procedimento de comparação de 10, 100, 1.000 e 10.000 dados, a Versão 2 executou a tarefa nos tempos médios de 15,6, 65,8, 1203 e 82497 milissegundos, respectivamente. Na Tabela 10 são relacionados os tempos cronometrados em cada uma das cinco repetições do teste e suas correspondentes médias.

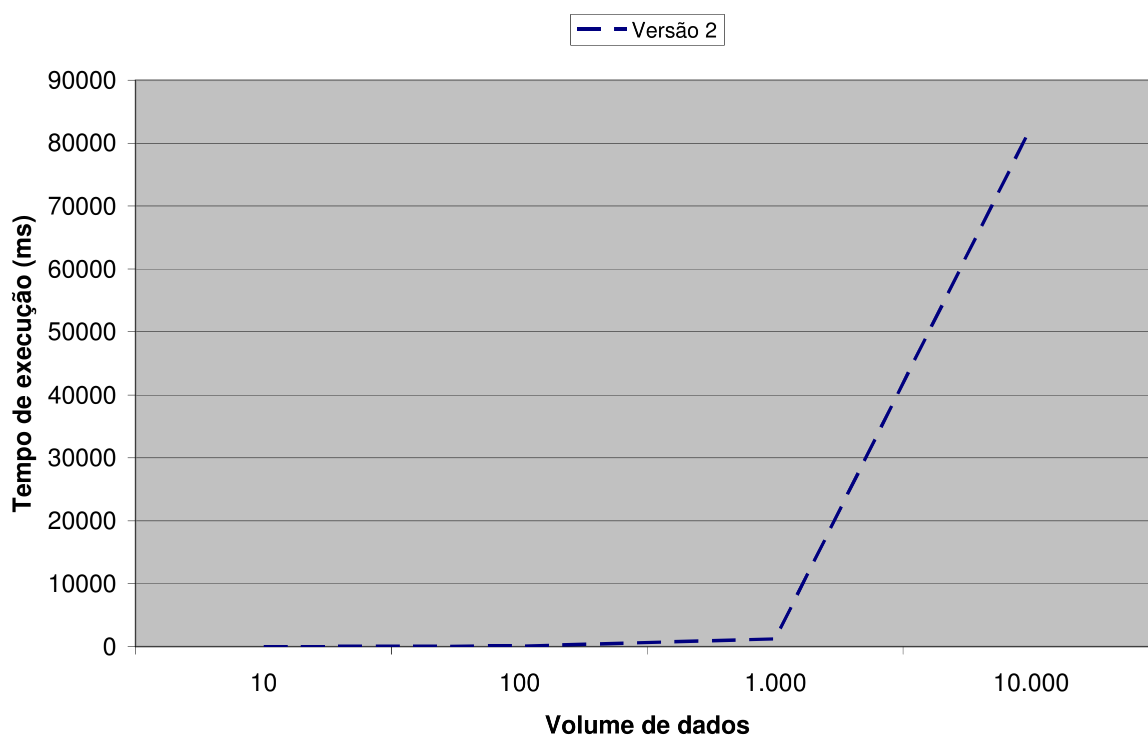
**Tabela 10 – Resultados obtidos no procedimento de comparação (Versão2)**

| Volume de Dados | Tempo 1 | Tempo 2 | Tempo 3 | Tempo 4 | Tempo 5 | Média |
|---|---|---|---|---|---|---|
| **10** | 16 | 16 | 16 | 15 | 15 | 15,6 |
| **100** | 78 | 63 | 62 | 63 | 63 | 65,8 |
| **1.000** | 1187 | 1219 | 1203 | 1203 | 1203 | 1203 |
| **10.000** | 82500 | 82516 | 82453 | 82688 | 82328 | 82497 |

tempo = em milissegundos

Em face destes resultados, o Gráfico 10 mostra o comportamento do procedimento de comparação de acordo com a quantidade de dados inseridos na Versão aqui tratada.

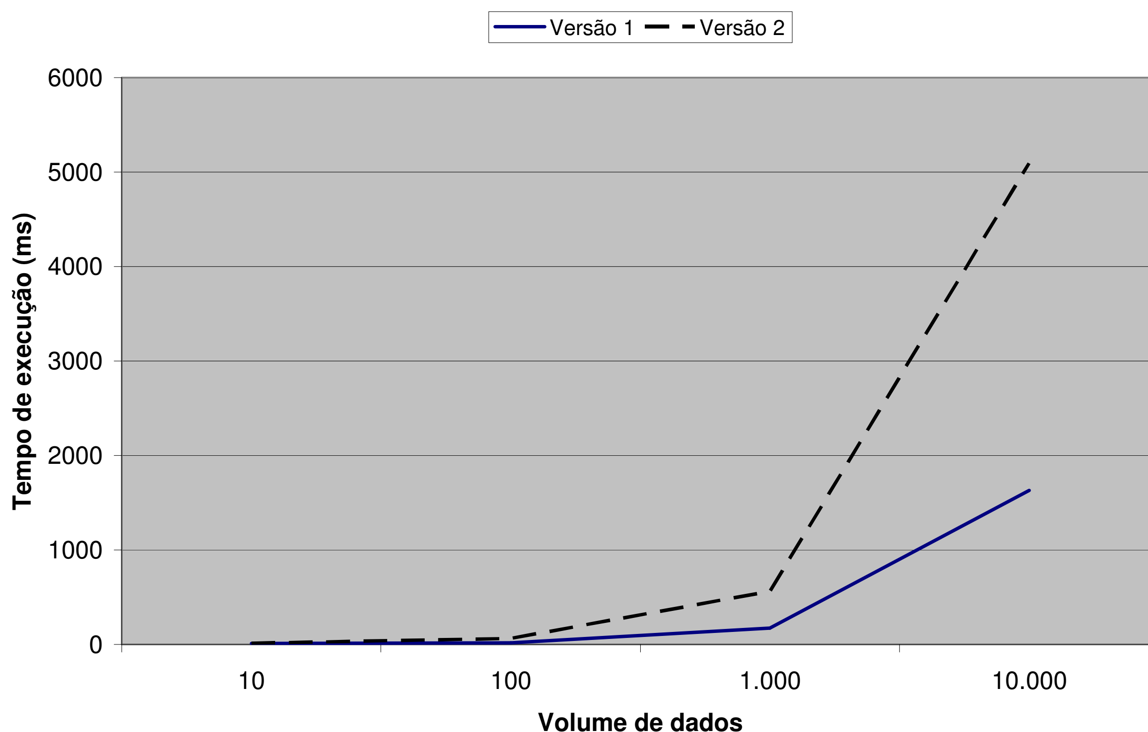
**Gráfico 10 - Comportamento do procedimento de comparação (Versão 2)**

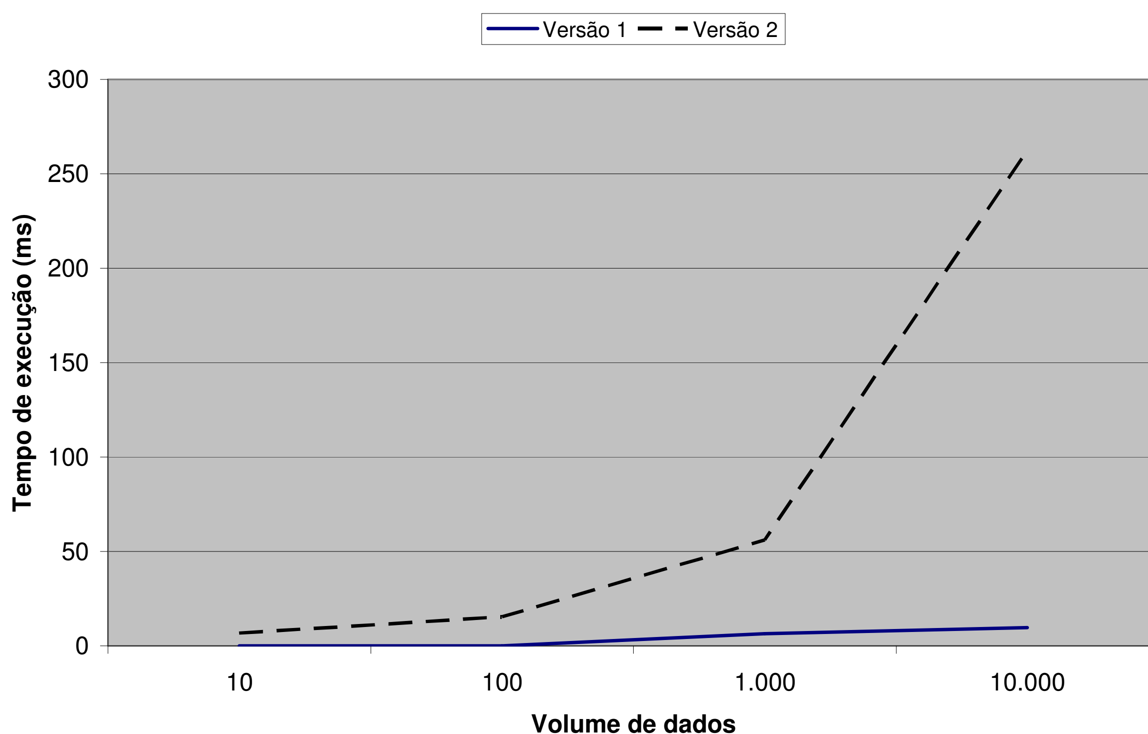



## 4.3 Análise e Comparação dos Resultados

Comparando-se os resultados obtidos pelas duas Versões, constata-se que a Versão 1 apresentou significativo melhor desempenho que a Versão 2 na execução das operações relativas a inserção e recuperação, a despeito de o volume de dados ser menor ou maior. As

performances das duas Versões para execução dos procedimentos de inserção e recuperação podem ser simultaneamente examinadas nos Gráficos 11 e 12, nos quais encontram-se denotada a superioridade da Versão 1 nessas tarefas.



Já na execução do procedimento de comparação, as Versões não apresentaram diferença significativa de performance, como se constata pela proximidade das linhas que compõe o Gráfico 13.

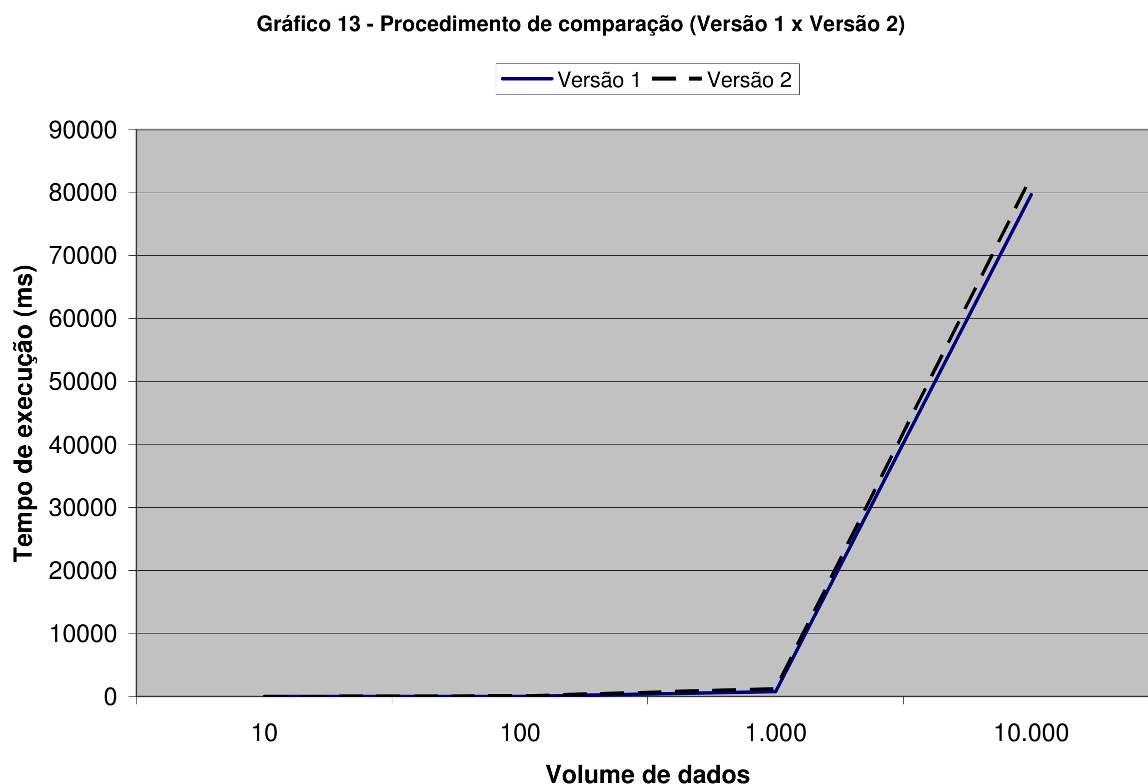



**Gráfico 13 - Procedimento de comparação (Versão 1 x Versão 2)**

Nos casos dos procedimentos de remoção e alteração, a Versão 2 – embora tenha se mostrado mais lenta na execução das tarefas com menor quantidade de dados (10, 100 e 1.000) – teve performance que superou a Versão 1 sempre que o volume de dados foi maior (10.000). Em tais situações, a diferença de tempo verificada entre as Versões foi de 6.187 ms, para remoção, e de 16.103,2 ms, para alteração. O melhor desempenho da Versão 2 nos casos de remoção e alteração, envolvendo maior quantidade de dados, pode ser visualizado nos Gráficos 14 e 15.

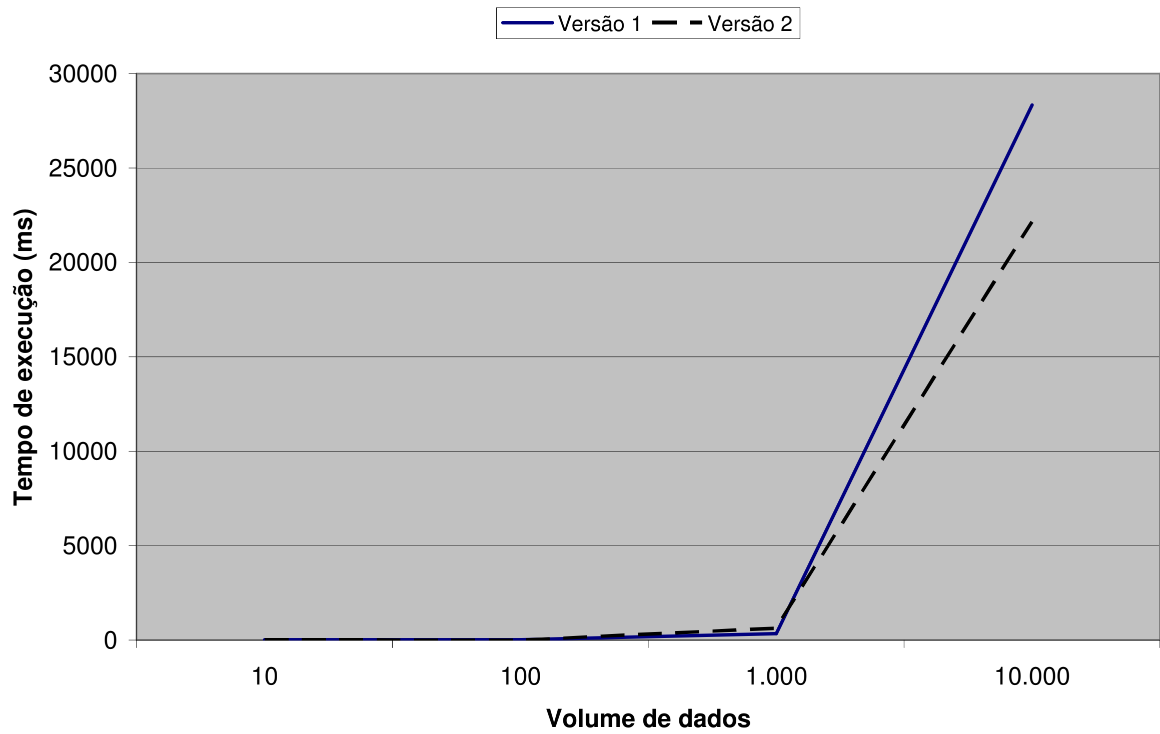


Gráfico 14 - Procedimento de remoção (Versão 1 x Versão 2)
Versão 1
Versão 2
30000
25000
20000
15000
10000
5000
0
Tempo de execução (ms)
10
100
1.000
10.000
Volume de dados

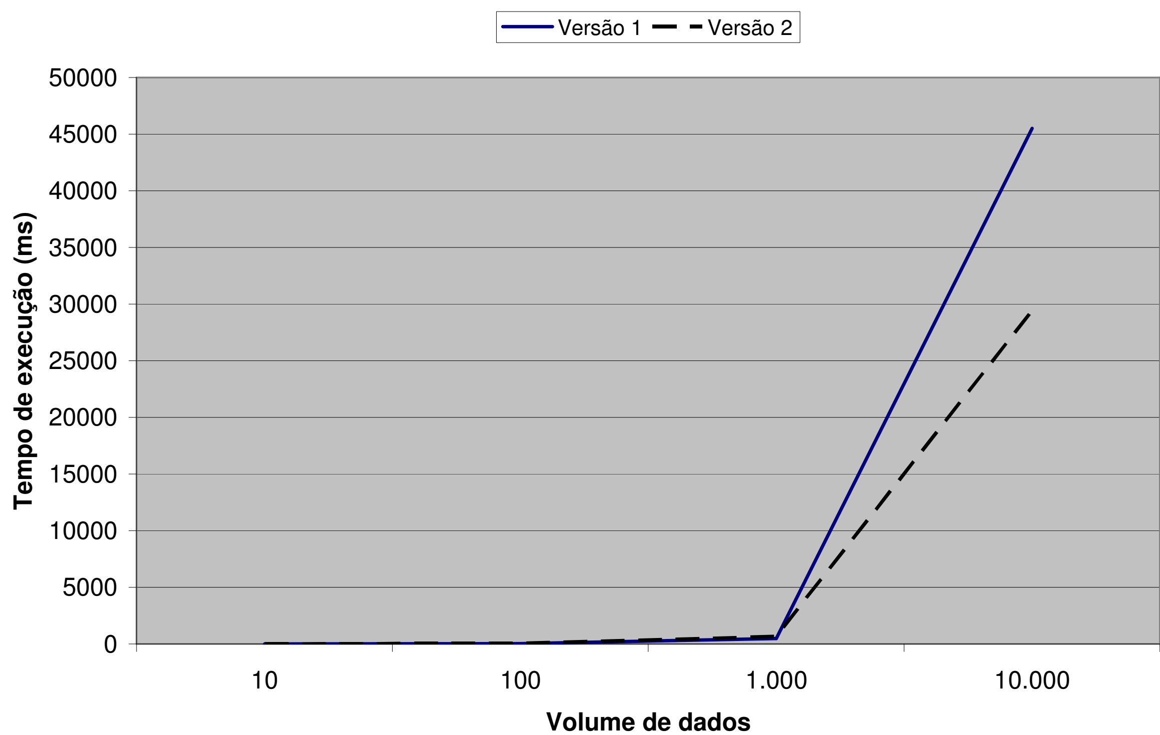
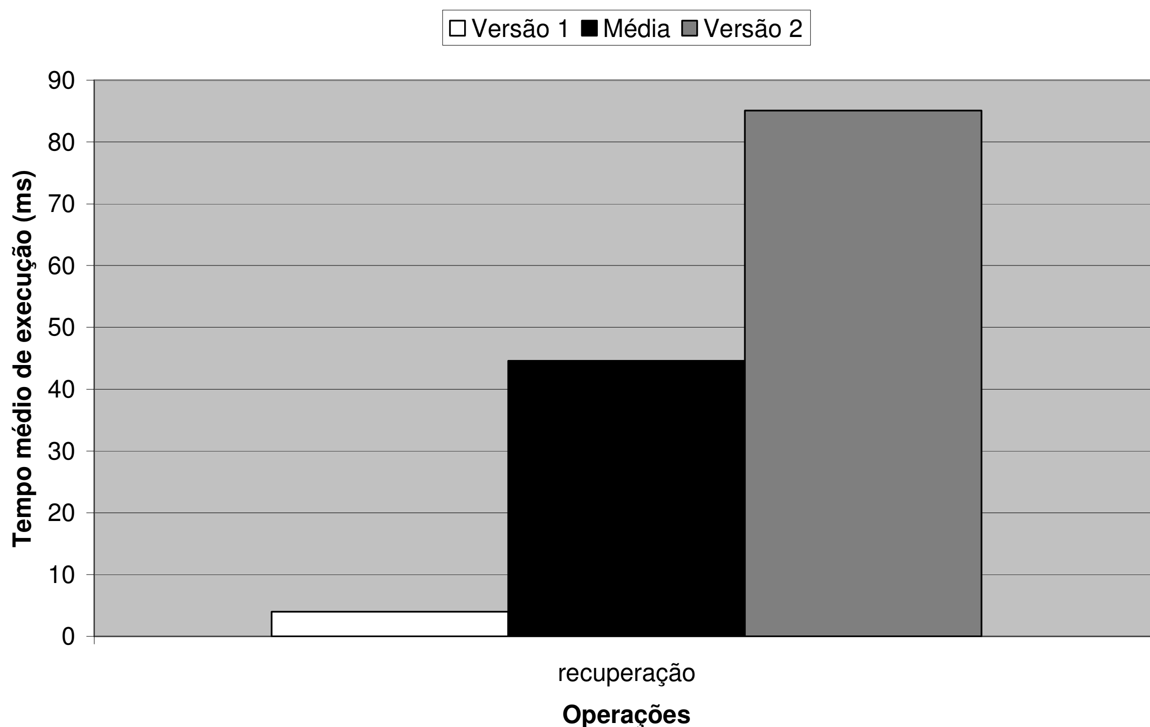


Gráfico 15 - Procedimento de alteração (Versão 1 x Versão 2)
Versão 1
Versão 2
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
Tempo de execução (ms)
10
100
1.000
10.000
Volume de dados

A comparação final das médias dos resultados obtidos nas avaliações das duas Versões, por operação solicitada, com as médias extraídas das médias das duas Versões, na mesma situação, é mostrada nos Gráficos 16 e 17, este referente às operações de inserção, comparação, remoção e alteração, e aquele referente às operações de recuperação.

**Gráfico 16 - Versão 1 x Versão 2: Comparação de performance (recuperação)**



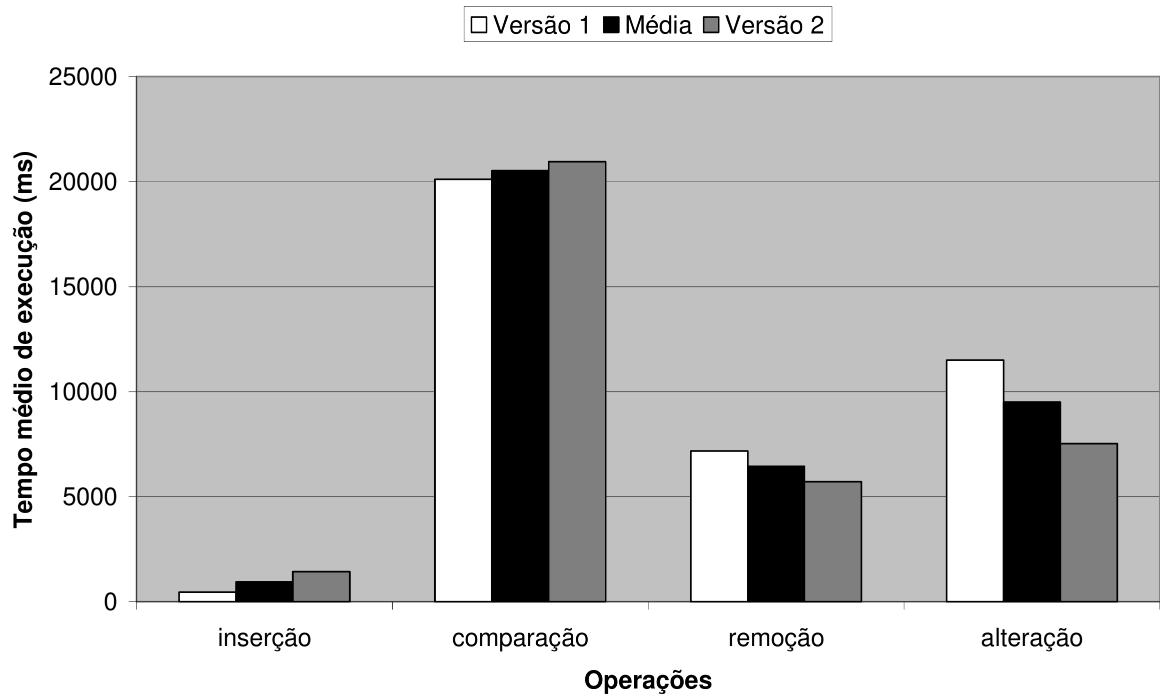


Gráfico 17 - Versão 1 x Versão 2: Comparação de performance (inserção, comparação, remoção e alteração)
Versão 1
Média
Versão 2
25000
20000
15000
10000
5000
0
Tempo médio de execução (ms)
inserção
comparação
remoção
alteração
Operações

# CONCLUSÃO

A comparação dos resultados das avaliações realizadas neste trabalho em um banco de dados, que numa versão teve os métodos implementados no próprio banco de dados (*stored procedure*) e na outra teve os métodos implementados na linguagem JAVA, levam às conclusões a seguir apontadas, uma vez consideradas as performances dessas duas alternativas.

Se o sistema de banco de dados for voltado para operações relativas a inserção e recuperação, independentemente do volume de dados solicitados, a implementação dos métodos no próprio banco de dados se apresenta como alternativa mais adequada em termos de performance.

Em se tratando de operações relativas a comparação de dados, também independentemente do volume deles, tanto a implementação dos métodos no próprio banco de dados quanto a implementação dos mesmos métodos na linguagem JAVA apresentam-se como alternativas similares, podendo ser qualquer delas adotada, sem prejuízo de performance.

No caso de o sistema de banco de dados ser mais solicitado para realização de operações de remoção ou alteração de pequenos volumes de dados (em torno de 1.000), a implementação dos métodos no próprio banco de dados se apresenta como alternativa mais eficiente em termos de performance.

Quando o sistema de banco de dados for mais requisitado para realização de operações de remoção ou alteração de maiores volumes de dados (em torno de 10.000), a implementação dos mesmos métodos na linguagem JAVA mostra-se como alternativa mais eficiente em termos de performance.

Como trabalhos futuros espera-se que a avaliação da performance aqui desenvolvida volte a ser aplicada a procedimentos mais complexos, com utilização de dados ainda mais próximos de um caso real, envolvendo mais de um recurso (inserção, recuperação, atualização, remoção e comparação) por procedimento avaliado.

# REFERÊNCIAS


BANCOS de dados orientados a objetos – conceitos fundamentais. Disponível em: <http://www.malima.com.br/article_read.asp?id=40>. Acesso em: 25 mar. 2008.

BOSCARIOLI, Clodis; BEZERRA, Anderson; BENEDICTO, Marcos de; DELMIRO, Gilliard.**Uma reflexão sobre Banco de Dados Orientados a Objetos.** Monografia de Conclusão de Curso **–** UNIOESTE, Cascavel, Paraná, 2006.

DEITEL, H. M.; DEITEL, P. J. **Java, como programar**. Tradução Carlos Arthur Lang Lisboa. 4.ed. Porto Alegre: Bookman, 2003, 1386 p. Título original: Java how to program.

EDELWEISS, Nina; GALANTE, Renata de M. **Banco de dados orientado a objetos e relacional-objeto**. [s.d.]. 46 p. Disponível em: <http://www.cultura.ufpa.br/clima/bdoo/galante.pdf>. Acesso em: 28 mar. 2008.

G. F. JÚNIOR, Olival de; PACHECO, Roberto C. S.; BARBOSA, Daniel M.; TODESCO, José L. **Abordando o uso da orientação a objetos em um sistema de *data warehouse*.** 2002. 13 p. Disponível em: <http://www.unifebe.edu.br/ftp/sistemas_de_informacao/ricardo_uriarte/oo1/bcd008.pdf >. Acesso em: 26 mar. 2008.

GONÇALVES, Glaura I.; PESENTE, Graziela R.; LIMA, Rafael. **Banco de dados orientado a objetos**. [s.d.]. Disponível em: <http://paginas.terra.com.br/informatica/arruda/Downloads/Artigos/artigo05/>. Acesso em: 24 mar. 2008.

INDRUSIAK, Leandro S. **Linguagem Java**, 1996. Disponível em: <http://www.inf.ufrgs.br/tools/java/introjava.pdf>. Acesso em: 07 mar. 2008.

LEITÃO, Daniel A.; BEZERRA, Edmo S. R.; ROLIM, Fabiano L. de S.; LOUREIRO, Janine de A.; MONTEIRO, Monique L. de B. **Plano de testes**. 2003. 9 p. Disponível em: <www.cin.ufpe.br/~dal/es/CAC_PlanoTestes.doc> Acesso em: 14 abr. 2008.

MARTIN, James; ODELL, James J. **Análise e projeto orientados a objeto**. Tradução José Carlos Barbosa dos Santos. São Paulo: Makron Books, 1995, 639 p. Título original: Object-Oriented Analysis and Design.

MENGUE, Fábio. **Curso de Java básico**, 2002, 35 p. Disponível em: <ftp://ftp.unicamp.br/pub/apoio/treinamentos/linguagens/java_basico.pdf>. Acesso em: 11 mar. 2008.

RICARTE, Ivan Luiz M. **Programação orientada a objetos: polimorfismo**, 2000. Disponível em: <http://www.dca.fee.unicamp.br/cursos/PooJava/polimorf/index.html>. Acesso em: 10 mar. 2008.

RICARTE, Ivan Luiz M. **Sistemas de bancos de dados orientados a objetos**, 1998, 41 p. Disponível em: <ftp://ftp.dca.fee.unicamp.br/pub/docs/ricarte/apostilas/mc_sbdoo.pdf>. Acesso em: 10 mar. 2008.

RUIZ, Flávio Henrique M.; GONÇALVES, Gustavo de S. **Desenvolvimento de sistema de banco de dados orientado a objetos com apoio da UML**. 2007. 107 f. Monografia (Graduação em Ciência da Computação) – Centro Universitário Eurípedes de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2007.

RUMBAUGH, James et al. **Modelagem e projetos baseados em objetos**. Tradução Dalton Conde de Alencar. Rio de Janeiro: Campus, 1994, 652 p. Título original: Object-oriented modeling and design.

SANCHES, André Rodrigo. **Fundamentos de armazenamento e manipulação de dados**. 2005. 6 p. Disponível em: <http://www.ime.usp.br/~andrers/aulas/bd2005-1/aula3.html>. Acesso em: 24 mar. 2008.

SANTOS JÚNIOR, João Benedito dos. **Linguagem de programação Java**, 2006, 67 p. Disponível em: <http://www.inf.pucpcaldas.br/~joao/cursos/javaxml/MateriaisApoio/ ApostilaLinguagemJAVA.pdf>. Acesso em: 08 mar. 2008.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S. **Sistema de Banco de Dados.** 3. ed. São Paulo: Makron Books, 1999, 778 p.

TAKAI, Osvaldo K.; ITALIANO, Isabel C.; FERREIRA, João E. **Introdução a banco de dados**. 2005. 124 p. Disponível em: <http://www.ime.usp.br/~jef/apostila.pdf>. Acesso em: 24 mar. 2008.