

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE
SISTEMAS

MURILO ANDRÉ DA SILVA

ESTUDO COMPARATIVO ENTRE AS LINGUAGENS PROCEDURAIS PL/SQL E
PL/PGSQL APLICADAS AOS BANCOS DE DADOS ORACLE 10G XE E
POSTGRESQL 8.4

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA

2011

MURILO ANDRÉ DA SILVA

**ESTUDO COMPARATIVO ENTRE AS LINGUAGENS PROCEDURAIS PL/SQL E
PL/PGSQL APLICADAS AOS BANCOS DE DADOS ORACLE 10G XE E
POSTGRESQL 8.4**

Trabalho de Diplomação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – CSTADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof Claudio Leones Bazzi.

MEDIANEIRA

2011



Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Curso Superior de Tecnologia em Análise e
Desenvolvimento de Sistemas



TERMO DE APROVAÇÃO

ESTUDO COMPARATIVO ENTRE AS LINGUAGENS PROCEDURAIS PL/SQL E PL/PGSQL APLICADAS AOS BANCOS DE DADOS ORACLE 10G XE E POSTGRESQL 8.4

Por

MURILO ANDRÉ DA SILVA

Este Trabalho de Diplomação (TD) foi apresentado às 8:30 h do dia 27 de setembro de 2011 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. O candidato foi argüido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. Msc. Claudio Leones Bazzi
UTFPR – *Campus* Medianeira
(Orientador)

Prof. Msc. Alan Gavioli
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Msc. Neylor Michel
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Juliano Rodrigo Lamb
UTFPR – *Campus* Medianeira
(Responsável pelas atividades de TCC)

RESUMO

SILVA, A. Murilo. ESTUDO COMPARATIVO ENTRE AS LINGUAGENS PROCEDURAIS PL/SQL E PL/PGSQL APLICADAS AOS BANCOS DE DADOS ORACLE 10G XE E POSTGRESQL 8.4. Trabalho de conclusão de curso (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Medianeira 2011.

Este trabalho tem como objetivo realizar um estudo comparativo entre as linguagens procedurais PL/SQL e PL/pgSQL dos bancos de dados Oracle 10g XE e PostgreSQL 8.4 respectivamente, que são linguagens comumente utilizadas por muitos desenvolvedores em aplicações, principalmente, de estrutura corporativa. Foram abordadas as diferenças de sintaxe e como é realizada a comunicação de cada linguagem com o Java através de suas bibliotecas. Também foi desenvolvida uma aplicação que funcionasse através de chamadas de procedimentos e funções criadas no banco de dados. Foram realizados testes de comparação de desempenho utilizando-se ferramentas de medição de tempo de resposta de cada banco de dados.

Palavras – Chaves: Sistemas Gerenciadores de Banco de Dados, *Procedural Language/Structured Query Language*, *Procedural Language/PostgreSQL Structured Query Language*.

RESUMO EM LÍNGUA ESTRANGEIRA

SILVA, A. Murilo. ESTUDO COMPARATIVO ENTRE AS LINGUAGENS PROCEDURAIS PL/SQL E PL/PGSQL APLICADAS AOS BANCOS DE DADOS ORACLE 10G XE E POSTGRESQL 8.4. Trabalho de conclusão de curso (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Medianeira 2011.

This research aims to conduct a comparative study between the procedural languages PL/SQL and PL/pgSQL of the Oracle 10g XE and PostgreSQL 8.4 database management systems respectively, which are languages commonly used by a great number of developers, mainly in corporative structured applications. Were approached the syntax differences and how the communication of the languages with Java is performed through its libraries. Also, was developed an application that worked by procedures and function callings created on the database. Were performed comparative performance tests utilizing database response time measuring tools.

Keywords: *Database Management Systems, Oracle, PostgreSQL, Java, Procedural Language/Structured Query Language, Procedural Language/PostgreSQL Structured Query Language.*

LISTAS DE QUADROS

Quadro 1 - Relação Cliente.....	16
Quadro 2 - Sintaxe básica de uma procedure	22
Quadro 3 - Estrutura de criação de uma função.....	24
Quadro 4 - Estrutura do bloco PL/pgSQL.....	27
Quadro 5 – Criação de Função no PostgreSQL.....	28
Quadro 6 – Função tcc_comparacao1 escrita em PL/SQL	30
Quadro 7 - Função tcc_comparacao1 convertida para PL/pgSQL.....	30
Quadro 8 - Procedure tcc_comparacao2.....	31
Quadro 9 - Comando para desfazer alterações no Oracle após uma exceção	32
Quadro 10 - Código de criação da tabela tb_calcado no banco Oracle	40
Quadro 11 - Código de criação da tabela tb_calcado no PostgreSQL.....	40
Quadro 12 - Código de criação da função fnc_listar_calcado no Oracle	42
Quadro 13 - Código de criação da função fnc_listar_calcado no Postgresql	42
Quadro 14 - Código de criação da <i>procedure</i> prc_obter_pk_calcado no Oracle.....	43
Quadro 15 - Código de criação da <i>procedure</i> prc_obter_pk_calcado no PostgreSQL	43
Quadro 16 - Código de criação da prc_atualizar_calcado no Oracle	44
Quadro 17 - Código de criação da <i>procedure</i> prc_atualizar_calcado no PostgreSQL	45
Quadro 18 - Código de criação da <i>procedure</i> prc_obter_por_id_calcado no Oracle .	46
Quadro 19 - Código de criação da <i>procedure</i> prc_obter_por_id_calcado no PostgreSQL	46
Quadro 20 - Código de criação da <i>procedure</i> prc_apaga_calcado no Oracle	47
Quadro 21 - Código de criação da <i>procedure</i> prc_apaga_calcado no PostgreSQL .	47
Quadro 22 - Código da classe Calcado	48

Quadro 23 - Classe conexão com o Oracle	50
Quadro 24 - Classe de conexão com PostgreSQL	50
Quadro 25 - Trecho do código do arquivo de propriedades spy.log	51
Quadro 26 - Código referente ao método atualizar calçado na classe CalcadoDAOOracle	55
Quadro 27 - Método listarCalcado na classe CalcadoDAOOracle.....	57
Quadro 28 - Método listarCalcado() na classe CalcadoDAOPostgreSQL	58
Quadro 29 - Código do método apagaCalcado na classe CalcadoDAOOracle.....	60

LISTAS DE TABELAS

Tabela 1 - Resultados dos testes realizados na função <i>fnc_listar_calçado</i>	72
Tabela 2 - Resultados dos testes realizados na <i>procedure prc_atualizar_funcionario</i>	73
Tabela 3 - Resultados dos testes realizados na <i>procedure prc_atualizar_cliente</i>	75
Tabela 4 - Resultados dos testes realizados na <i>procedure prc_apagar_funcionario</i>	77
Tabela 5 - Tabela de resultados específicos.....	78

LISTAS DE FIGURAS

Figura 1 - Bloco PL/SQL e sua estrutura básica	20
Figura 2 - Modelo do ambiente PL/SQL.....	21
Figura 3 - Tipos de blocos PL/SQL.....	22
Figura 4 - Funcionamento de uma compilação ou execução de um programa Java.	34
Figura 5 - Estrutura de ligação entre o Java e os Bancos de Dados.....	36
Figura 6 - Diagrama de Classes da Aplicação.....	39
Figura 7 - Diagrama Entidade Relacionamento descrevendo a aplicação	41
Figura 8 - Estrutura de pacotes da aplicação	49
Figura 9 - Tela principal da aplicação.....	51
Figura 10 - Inserção de Calçado	52
Figura 11 - Tela de calçados no momento de uma consulta.....	53
Figura 12 - Tela de calçado no momento de uma alteração	53
Figura 13 - Diagrama de sequencia para a inserção de calçados	54
Figura 14 - Diagrama de Sequência para listagens de calçado	56
Figura 15 - Diagrama de Sequência relativo a uma exclusão de calçado	59
Figura 16 - Tela de Venda.....	60
Figura 17 - Tela de adição de calçados na venda	61
Figura 18 - Tela de adição de calçados na venda no momento que a quantidade é informada.	61
Figura 19 - Tela de venda com todos os dados inseridos prestes a ser realizada....	62
Figura 20 - Diagrama de Sequencia referente a realização de uma venda.....	63
Figura 21 - Trecho onde é chamado o procedimento atualizarVenda()	63
Figura 22 - Trecho do diagrama exibindo a execução do método atualizaEstoque()	64
Figura 23 - Trecho do diagrama exibindo o chamamento do método insereCalçado_Venda()	64

Figura 24 - Tela de venda exibindo todas as vendas registradas no banco.....	65
Figura 25 - Diagrama de sequencia relativo a venda.....	66
Figura 26 - Tela de venda com uma venda prestes a ser atualizada	67
Figura 27 - Diagrama de sequencia descrevendo como é feita uma atualização de venda no Oracle.....	68
Figura 28 - Estrutura dos testes de desempenho	69
Figura 29 - Tela do SQL Profiler v0.3 com uma requisição capturada	70

LISTAS DE ABREVIATURAS

API	<i>Application programming interface</i>
DBA	<i>Database Administrator</i>
DDL	<i>Data Definition Language</i>
DML	<i>Data Manipulation Language</i>
IBM	<i>International Business Machines</i>
iSQL	<i>Internet SQL Command Line</i>
J2ME	<i>Java 2 Mobile Edition</i>
J2SE	<i>Java 2 Standard Edition</i>
JDBC	<i>Java Data Base Connectivity</i>
JDBC	<i>Java Database Connectivity</i>
JDK	<i>Java Development Kit</i>
JRE	<i>Java Runtime Edition</i>
JVM	<i>Java Virtual Machine</i>
ODBC	<i>Open Data Base Connectivity</i>
PHP	<i>PHP: Hypertext Preprocessor</i>
PL	<i>Procedural Language</i>
PL/pgSQL	<i>Procedural Language/PostgreSQL Structured Query Language</i>
PL/SQL	<i>Procedural Language/Structured Query Language</i>
SDK	<i>Standard Development Kit</i>
SQL	<i>Structured Query Language</i>
SGBD	<i>Sistema Gerenciador de Banco de Dados</i>
XE	<i>Express Edition</i>
XML	<i>Extensible Markup Language</i>

SUMÁRIO

1	INTRODUÇÃO	13
1.1	OBJETIVO GERAL	13
1.2	OBJETIVOS ESPECÍFICOS	13
1.3	JUSTIFICATIVA	14
1.4	ESTRUTURA DO TRABALHO	14
2	SISTEMAS GERENCIADORES DE BANCO DE DADOS	16
2.1	MODELO RELACIONAL	16
2.2	LINGUAGEM SQL	16
2.3	ORACLE	18
2.4	ORACLE DATABASE 10G EXPRESS EDITION	18
2.5	FERRAMENTAS PARA A MANIPULAÇÃO DOS DADOS	19
2.6	A LINGUAGEM PL/SQL	19
2.7	BLOCOS NA LINGUAGEM PL/SQL	20
2.7.1	Tipos de Blocos	21
2.8	POSTGRESQL	24
2.9	POSTGRESQL 8.4	25
2.10	A LINGUAGEM PL/PGSQL	26
2.10.1	Funções	27
2.11	COMPARAÇÃO ENTRE AS LINGUAGENS PL/SQL E PL/PGSQL	28
2.12	EXEMPLOS DE DIFERENÇAS:	29
2.12.1	Conversão de funções	29
2.12.2	Diferença em recursos nativos	31
2.12.3	Exceções	31
2.12.4	Commit	32
3	JAVA	33
3.1	UTILIZANDO ORACLE E POSTGRESQL EM APLICAÇÕES JAVA	35
3.1.1	<i>Drivers</i> JDBC Oracle	36
3.1.2	<i>Drivers</i> JDBC PostgreSQL	37
4	ESTUDO DE CASO	38
4.1	SOFTWARES UTILIZADOS	38

4.2	DESENVOLVIMENTO DA APLICAÇÃO.....	38
4.3	DESENVOLVIMENTO	39
4.3.1	Criação da estrutura do banco de dados.....	39
4.3.2	Desenvolvimento da aplicação através do Java.....	47
5	TESTES DE DESEMPENHO	69
5.1	TESTES DA FUNÇÃO FNC_LISTAR_CALCADO()	70
5.1.1	Testes da função FNC_LISTAR_CALCADO – Oracle.....	70
5.1.2	Testes da função FNC_LISTAR_CALCADO – PostgreSQL.....	71
5.1.3	Resultados dos testes na função fnc_listar_calcado	71
5.2	TESTES DA PROCEDURE PRC_ATUALIZAR_FUNCIONARIO	72
5.2.1	Testes da <i>procedure</i> PRC_ATUALIZAR_FUNCIONARIO – Oracle	72
5.2.2	Testes da <i>procedure</i> PRC_ATUALIZAR_FUNCIONARIO – PostgreSQL	73
5.2.3	Resultados dos testes na <i>procedure</i> prc_atualizar_funcionario.....	73
5.3	TESTES DA PROCEDURE PRC_ATUALIZAR_CLIENTE()	74
5.3.1	Teste PRC_ATUALIZAR_CLIENTE - ORACLE.....	74
5.3.2	Teste PRC_ATUALIZAR_CLIENTE – PostgreSQL	74
5.3.3	Resultados dos testes na <i>procedure</i> prc_atualizar_cliente.....	75
5.4	TESTES DA PROCEDURE PRC_APAGA_FUNCIONARIO	75
5.4.1	Teste PRC_APAGA_FUNCIONARIO – Oracle	75
5.4.2	Teste PRC_APAGAR_FUNCIONARIO – POSTGRESQL.....	76
5.4.3	Resultados dos testes na <i>procedure</i> prc_apagar_funcionario	77
6	CONSIDERAÇÕES FINAIS	78
6.1	CONCLUSÃO	78
6.2	TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO.....	79
	REFERÊNCIAS BIBLIOGRÁFICAS.....	80

1 INTRODUÇÃO

Na fase inicial da informática, os dados ou informações eram dependentes de aplicações responsáveis por manipular esses conjuntos de dados. Isso ocasionou uma limitação por parte dos utilizadores já que as estruturas de dados eram definidas apenas pelos programadores. Por isso, qualquer alteração na estrutura da informação, como por exemplo, incluir ou retirar campos em um arquivo na base de dados, implicava que os programadores tivessem de alterar os programas de aplicação que operavam esses dados. Desta dificuldade foi então que surgiram os Sistemas Gerenciadores de Banco de Dados (ESPACOINFO.NET, 2011).

Um SGBD - Sistema de Gerenciamento de Banco de Dados é uma coleção de programas que permitem ao usuário definir, construir e manipular bases de dados para as mais diversas finalidades. Entre os mais famosos estão o Oracle, o PostgreSQL, Microsoft SQL Server e o MySQL. Para manipulação dos SGBDs são utilizadas linguagens como a SQL (NOVELLI, 2006).

O foco deste trabalho é o estudo sobre os bancos de dados Oracle e PostgreSQL, abordando suas linguagens de programação procedurais, o PL/SQL e o PL/pgSQL. O estudo envolveu a integração das linguagens com o Java através de uma aplicação abrangendo testes comparativos de desempenho.

1.1 OBJETIVO GERAL

Desenvolver um estudo comparativo sobre os SGBD's Oracle 10g XE e PostgreSQL 8.4 e suas linguagens procedurais PL/SQL e PL/pgSQL respectivamente, objetivando realizar a comunicação e integração dessas linguagens com a linguagem de programação Java, via JDBC.

1.2 OBJETIVOS ESPECÍFICOS

- Realizar um referencial teórico sobre o Banco de Dados Oracle 10g XE e sobre o Banco de Dados PostgreSQL bem como suas linguagens procedurais PL/SQL e PL/pgSQL respectivamente;
- Realizar um estudo sobre a integração entre a linguagem Java e as linguagens PL/SQL e PL/pgSQL através do desenvolvimento de uma aplicação utilizando Java;

- Utilizar o *driver* JDBC juntamente com a linguagem Java, para fazer a comunicação com o banco de dados acessando procedimentos e funções criados nas linguagens PL/SQL e PL/pgSQL;
- Realizar testes de desempenho na aplicação utilizando a ferramenta SQL Profiler 0.3 e o P6Spy;
- Realizar um comparativo entre as linguagens procedurais abordadas no trabalho.

1.3 JUSTIFICATIVA

Os Bancos de Dados, além de manterem todo o volume de dados organizado, também executam tarefas e comandos que são previamente programadas por um DBA – *Data Base Administrator*, que define os serviços a serem realizados pelo sistema, sendo que a sua capacidade influi diretamente no desempenho do banco de dados (MENEZES et al, 2008).

Existem SGBD's voltados para aplicações de pequeno a grande porte e independente do tamanho do sistema ele deverá sempre operar de forma otimizada e satisfatória (PEREZ, F. C. LUÍS, 2010).

A utilização de linguagens procedurais de banco de dados como PL/SQL e PL/pgSQL trazem um ganho de performance já que são passados comandos em blocos ao invés de forma individual como no SQL comum (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2011).

Um estudo com o objetivo de comparar essas duas linguagens é válido no sentido de trazer um aprofundamento sobre dois dos principais bancos de dados disponíveis no mercado (RIBEIRO, 2010).

1.4 ESTRUTURA DO TRABALHO

O trabalho foi dividido em seis capítulos de que abordarão os seguintes temas:

O capítulo dois fala sobre os sistemas gerenciadores de banco de dados, descrevendo um pouco de sua história, seus componentes. Este capítulo também trata do modelo de dados adotado, o modelo relacional. Também fala sobre os bancos utilizados neste trabalho, o Oracle e o PostgreSQL e por último faz um comparativo entre as linguagens procedurais PL/SQL e PL/pgSQL. Trás exemplos

de suas diferenças e como são feitas as conversões de código entre as duas linguagens.

O capítulo seguinte fala sobre a linguagem de programação Java e como é feita a integração da mesma com as linguagens PL/SQL e PL/pgSQL.

O capítulo quatro trás um estudo de caso envolvendo uma aplicação referente a uma loja de calçados onde são realizados chamamentos de procedimentos e funções criadas nos bancos de dados estudados.

No capítulo cinco são demonstrados os testes de desempenho na aplicação bem como seus resultados.

No sexto capítulo são apresentadas as conclusões deste trabalho.

2 SISTEMAS GERENCIADORES DE BANCO DE DADOS

Segundo Helland (2009, p. 714), um SGDB é um sistema designado para fornecer acesso a dados de forma controlada e gerenciável. Por permitir a definição da estrutura dos dados de forma separada, um SGBD liberta a aplicação de muitos detalhes onerosos na manutenção e no fornecimento de seus dados.

Os Bancos de Dados, além de manterem todo o volume de dados organizado, também executam tarefas e comandos que são previamente programadas por um DBA – *Data Base Administrator*, que é o responsável pela manutenção e define os serviços a serem realizados pelo sistema (MENEZES et al, 2008).

2.1 MODELO RELACIONAL

Segundo Eembley (2009, p. 2372), o modelo relacional (utilizado neste trabalho) descreve os dados como relações nomeadas compostas chaves e valores. Por exemplo, o código de um cliente pode se relacionar com o seu nome e seu endereço em uma relação chamada Cliente.

No Quadro 1 é apresentada a relação Cliente, onde existem os pares chave-valor, por exemplo “código do cliente – 11111”. Esse pares são conhecidos como tuplas, ou chamados de registros, da relação cliente.

Cliente	
codigo do cliente	11111
nome do cliente	Pat
endereço do cliente	Maple 12

Quadro 1 - Relação Cliente
Fonte: Adaptado de Eembley (2009, p. 2372)

Ainda segundo Eembley (2009, p. 2372), normalmente as relações do modelo relacional são vistas como tabelas e estas juntas formam uma base de dados relacional. Além das suas estruturas, as tabelas no modelo relacional também possuem *Constraints* (*Not null, check, foreign key, unique key e primary key*).

2.2 LINGUAGEM SQL

A *Structured Query Language* ou SQL é a linguagem de consulta em banco de dados mais utilizada no mundo. Foi desenvolvida no *IBM Research Laboratories*

na década de 1970, baseada no modelo relacional definido por E. F. Codd em 1970, que suporta a recuperação, manipulação e administração de dados armazenados em tabelas (CHAMBERLIN, 2009)

Atualmente se encontra na versão SQL:2008 e foi padronizada em 1986 pela *American National Standards Institute*.

De acordo com Ramakrishnan(2009, p.131-132), a linguagem SQL possui alguns aspectos principais:

- Linguagem de manipulação de dados ou DML: Esse subconjunto do SQL permite os usuários realizar consultas para inserir, apagar ou modificar linhas.
- Linguagem de definição de dados ou DDL: Esse subconjunto do SQL suporta a criação, exclusão e modificação de definições para as tabelas e *views*. Restrições de integridades podem ser definidas nas tabelas, seja no momento que a tabela está sendo criada ou mais tarde.
- *Triggers* e Restrições de Integridade Avançadas: A partir da quarta revisão da linguagem, chamada de SQL:1999 ou também conhecida como SQL3, foi incluído o suporte para *triggers*. Elas são ações executadas pelo SGBD toda vez que alguma mudança no banco de dados atinge certos requisitos definidos por ela.
- SQL Dinâmico e Embutido: O SQL Embutido, ou *Embedded* no inglês, permite que a SQL seja chamada de outra linguagem, como por exemplo a linguagem C. O SQL dinâmico permite que as consultas sejam construídas e executadas em tempo de execução.
- Execução Cliente-Servidor e Acesso a Banco de Dados Remoto: Esses comandos controlam como uma aplicação cliente pode se conectar com servidor de banco de dados SQL ou acessar os dados de uma base dados em uma rede.
- Gerenciamento de transações: Vários comandos que permitem um usuário controlar explicitamente os aspectos de como uma transação podem ser executados.
- Segurança: O SQL fornece mecanismos para controlar o acesso de usuários em objetos de dados como tabelas e consultas.

- Características avançadas: o SQL permite a inclusão de orientação objeto, *queries* recursivas, *queries* de suporte a decisão, *data mining*, dados espaciais e gerenciamento de dados XML ou texto.

2.3 ORACLE

Oracle é um sistema de banco de dados que surgiu no final dos anos 70, quando Larry Ellison vislumbrou uma oportunidade que outras companhias não haviam percebido quando encontrou uma descrição de um protótipo funcional de um banco de dados relacional e descobriu que nenhuma empresa tinha se empenhado em comercializar essa tecnologia. Ellison e os co-fundadores da Oracle Corporation, Bob Miner e Ed Oates, perceberam que havia um tremendo potencial de negócios no modelo de banco de dados relacional tornando assim a maior empresa de software empresarial do mundo. Com faturamento anual superior a 10,8 bilhões de dólares, a empresa oferece seus produtos de bancos de dados, ferramentas e aplicativos, bem como serviços relacionados de consultoria, treinamento e suporte (SOUZACE, 2011).

O Oracle 9i foi pioneiro no suporte ao modelo web. O Oracle 10g, mais recente, se baseia na tecnologia de *grid*¹. Recentemente foi lançado o Oracle 11g que veio com melhorias em relação ao Oracle 10g.

Além da base de dados, a Oracle desenvolve uma *suíte* de desenvolvimento chamada de *Oracle Developer Suite*, utilizada na construção de programas de computador que interagem com a sua base de dados.

2.4 ORACLE DATABASE 10G EXPRESS EDITION

O *Oracle Database 10g Express Edition* ou *Oracle Database XE* é uma versão reduzida e grátis baseada no código do *Oracle Database 10g Release 2*.

A Oracle indica o *Database XE* para:

- Desenvolvedores trabalhando com PHP, Java, .NET, XML e aplicações *Open Source*.
- DBAs que precisam de um banco de dados livre, simples para treinamento.

¹ Segundo Ian Foster (2009), um Grid é um sistema que coordena recursos que não estão sujeitos a um controle centralizado, usa protocolos e interfaces normalizadas, abertas e de uso genérico para garantir a interoperabilidade entre sistemas diferentes e fornece serviços com uma qualidade não trivial de modo a satisfazer as necessidades dos utilizadores mesmo que estas sejam complexas.

- Produtores independentes de *softwares* e vendedores de *hardware* que querem uma distribuição de banco de dados livre de custo.
- Instituições educacionais e estudantes que precisam de um banco de dados para estudo.

De acordo com a Oracle Corporation, através do *Oracle Database XE* é possível criar aplicações com boa infraestrutura e se necessário é possível realizar uma migração sem complexidade e de baixo custo.

2.5 FERRAMENTAS PARA A MANIPULAÇÃO DOS DADOS

A Oracle fornece três aplicativos para a manipulação de dados: O *Internet SQL Command Line (iSQL)*, que é utilizado via *browser* e executa comandos SQL e PL/SQL. O *SQL Command Line*, também conhecido como *SQL Plus*. E também o *SQL Developer*, este voltado para ambiente *Desktop*.

2.6 A LINGUAGEM PL/SQL

A sigla PL/SQL - *Procedural Language Extensions to the Structured Query Language* ou Extensões de Linguagens Procedurais para SQL. Essa linguagem foi desenvolvida pela Oracle para superar algumas relações da SQL e para fornecer uma solução mais programática para aqueles que buscam uma gama de recursos maior e mais pratica em suas aplicações no banco de dados (FEUERSTEIN, PRIBYL, 2009).

Feuerstein e Pribyl (2009, p. 3) afirmam que a linguagem PL/SQL é uma linguagem procedural de fácil interpretação, com uma estrutura e palavras chaves que expressam com clareza o objetivo do código, indicada tanto para programadores experientes ou para os iniciantes. Ela possui uma grande portabilidade no sentido de que se uma *procedure* ou função for escrita em um simples *notebook*, ela pode ser passada com facilidade para uma rede corporativa.

A linguagem PL/SQL é bastante versátil no que diz respeito ao tipo de aplicação que ela pode ser voltada. Pode ser utilizada no Java com JDBC (Java Database Connectivity), no Visual Basic com ODBC (Open Data Base Connectivity) ou Delphi, C++, entre outros (FEUERSTEIN, PRIBYL, 2009).

A linguagem PL/SQL combina o poder da manipulação de dados do SQL com o poder de processamento das linguagens procedurais. Através dela pode-se

controlar o fluxo do programa através de comandos como IF e LOOP. Ela apresenta ainda declaração de variáveis, definição de *procedures*, funções e apresentação de erros (Oracle, 2005).

2.7 BLOCOS NA LINGUAGEM PL/SQL

De acordo com a Oracle (2005), Toda unidade PL/SQL compreende um ou mais blocos. Esses blocos podem ser inteiramente separados ou aninhados um dentro do outro. Um bloco agrupa declarações relacionadas. Podem-se colocar declarações perto do local onde serão utilizadas, como se formassem um subprograma dentro de um programa maior. As declarações são referentes apenas ao bloco e deixam de existir quando esse bloco se completa, ajudando assim a evitar variáveis e *procedures* soltas na memória.

A Figura 1 mostra que um bloco PL/SQL possui três partes básicas: O *Declare*, que a parte declarativa, o BEGIN e o END, que compõem a parte executável e o *Exception* que é voltado para o tratamento de exceções. Apenas a parte executável é necessária. A parte declarativa é escrita primeiro e é o local aonde são definidos os tipos, as variáveis e os itens similares. Esses itens são manipulados na parte executável. Exceções levantadas durante a execução podem ser corrigidas com a parte de tratamento de erros.

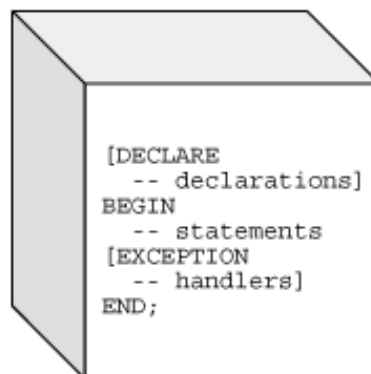


Figura 1 - Bloco PL/SQL e sua estrutura básica
Fonte: Oracle (2005)

Sem a utilização da linguagem PL/SQL, o Oracle precisaria processar os comandos SQL um de cada vez. Programas que fazem múltiplas requisições SQL fazem com que múltiplas chamadas sejam feitas à base de dados, resultando em excesso de performance e tráfego na rede (Oracle, 2005).

De acordo com a Oracle, o mecanismo PL/SQL faz a análise da sintaxe do bloco enviado ao servidor para definir quais instruções são do tipo SQL e quais instruções são do tipo PL/SQL. Depois ele delega cada instrução para o seu respectivo executor conforme visto na Figura 2:

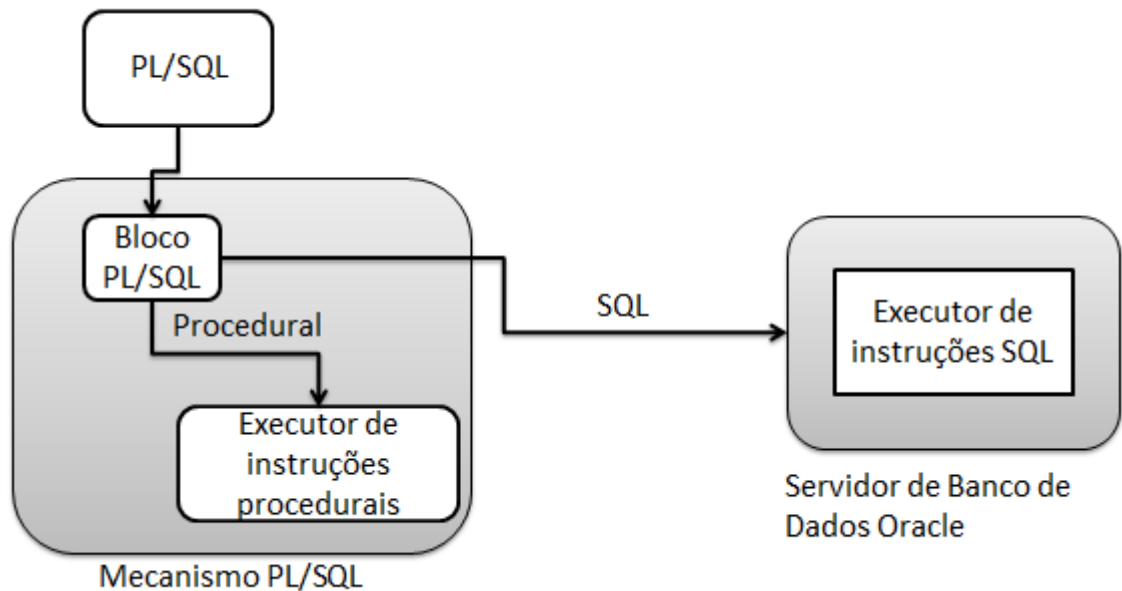


Figura 2 - Modelo do ambiente PL/SQL
 Fonte: Oracle - Adaptado (2005)

2.7.1 Tipos de Blocos

Existem dois tipos de construções PL/SQL disponíveis: Os blocos anônimos e os subprogramas.

Os blocos anônimos são blocos sem nome que são declarados em um ponto do aplicativo onde devem ser utilizados e são passados para o mecanismo PL/SQL para serem executados.

Os subprogramas são blocos PL/SQL nomeados que podem assumir parâmetros e podem ser chamados. Pode-se declará-los como procedimentos ou como funções, sendo que geralmente, utiliza-se um procedimento para desempenhar uma ação e uma função para calcular um valor. Uma função é similar a um procedimento, exceto que uma função deve retornar um valor (Oracle, 2000). A Figura 3 ilustra os blocos.

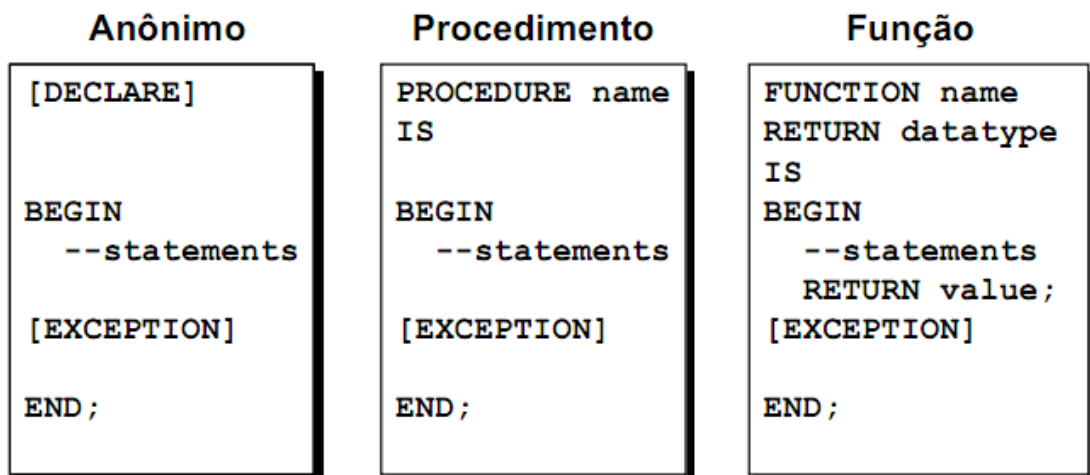


Figura 3 - Tipos de blocos PL/SQL
Fonte: Oracle (2000)

2.7.1.1 Procedure

Goya (2004) diz que uma *procedure* é um bloco PL/SQL nomeado que pode aceitar argumentos e pode ser chamada por um programa, uma sessão SQL ou uma *trigger*. A grande vantagem sobre um bloco PL/SQL anônimo é que pode ser compilado e armazenado no banco de dados como um objeto de *schema*². Graças a essa característica as *procedures* são de fácil manutenção, o código é reutilizável e permitem trabalhar com módulos de programa.

Segundo Goya(2004), a sintaxe básica de uma *procedure* é:

```
CREATE [OR REPLACE] PROCEDURE [schema.]nome_da_procedure
[(parâmetro1 [modo1] tipodedado1,
  parâmetro2 [modo2] tipodedado2,
  ...)]
IS|AS
Bloco PL/SQL
```

Quadro 2 - Sintaxe básica de uma procedure
Fonte: Goya (2004).

* [...] – indica não obrigatoriedade

²No banco Oracle, associados a cada usuário da base de dados existe um schema. Ele consiste uma coleção de objetos de schema como por exemplo: tabelas, veiw's, sequencias, sinonimos, indices, clusters, procedures, funções e pacotes. (Oracle, 1999)

em que:

- REPLACE - indica que caso a *procedure* exista ela será eliminada e substituída pela nova versão criada pelo comando;
- BLOCO PL/SQL - inicia com uma cláusula BEGIN e termina com END ou END nome_da_procedure;
- NOME_DA_PROCEDURE - indica o nome da *procedure*;
- PARÂMETRO - indica o nome da variável PL/SQL que é passada na chamada da *procedure* ou o nome da variável que retornará os valores da *procedure* ou ambos. O que irá conter em parâmetro depende de MODO;
- MODO - Indica que o parâmetro é de entrada (IN), saída (OUT) ou ambos (IN OUT). É importante notar que IN é o modo *default*, ou seja, se não for explicitado nada o modo do parâmetro será, automaticamente, IN;
- TIPODEDADO - indica o tipo de dado do parâmetro. Pode ser qualquer tipo de dado do SQL ou do PL/SQL. Pode usar referências como %TYPE, %ROWTYPE ou qualquer tipo de dado escalar ou composto. Não é possível fazer qualquer restrição ao tamanho do tipo de dado neste ponto.
- IS|AS - a sintaxe do comando aceita tanto IS como AS. Por convenção utiliza-se IS na criação de *procedures* e AS quando estiver criando pacotes.
- BLOCO PL/SQL - indica as ações que serão executadas por aquela *procedure*.

2.7.1.2 Funções

Funções e *procedures* são elementos parecidos, com a diferença que uma função possui uma cláusula de retorno ao invés de um argumento OUT ou IN OUT. Diferente de uma chamada de *procedure* que é executada de forma autônoma, a chamada de uma função só pode existir como parte de uma sentença executável, como um elemento em uma expressão ou um valor atribuído como padrão em uma declaração de variável (FEUERSTEIN, PRIBYL, 2009).

O comando *RETURN* é responsável por retornar o valor de uma função. O valor do retorno será convertido para o tipo especificado no começo da função. Uma função pode possuir várias instruções *RETURN* no seu corpo, mas apenas uma é executada (KALLAS, 2008).

Segundo Goya, a estrutura de criação de uma função corresponde a apresentada no Quadro 3:

```
CREATE [OR REPLACE] FUNCTION nome_da_função
[( parameter1 [ modo1] tipodedado1,
parameter2 [ modo2] tipodedado2,
. . .)]
RETURN tipo_de_dado
IS|AS
Bloco PL/SQL;
```

Quadro 3 - Estrutura de criação de uma função
Fonte: Goya (2011)

* [...] – indica não obrigatoriedade

2.8 POSTGRESQL

O PostgreSQL é um sistema gerenciador de banco de dados objeto-relacional baseado no POSTGRES *Version 4.2* desenvolvido na Universidade da Califórnia em Berkeley no departamento de ciência da computação. O PostgreSQL foi pioneiro em vários conceitos que se tornaram disponíveis em sistemas comerciais muito tempo depois (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2009).

É considerado *open-source*, sendo originário do código criado em Berkeley, dando suporte a maioria do padrão SQL, oferecendo recursos modernos, tais como a utilização de

Queries complexas;

Chaves estrangeiras;

Triggers;

Views;

Integridade de transação e Controle de concorrência multiversão. Fornece suporte ainda, a extensões realizadas pelo usuário em muitas formas, como por exemplo a adição de novos tipos de dados, Funções, Operadores, funções agregadas, métodos de indexação e linguagem procedurais.

Devido sua licença, o PostgreSQL pode ser utilizado, modificado e distribuído livre de custos para qualquer propósito, seja ele comercial, privado ou acadêmico (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2009).

O PostgreSQL permite que as funções definidas pelo usuário sejam escritas em outras linguagens além de SQL e C. Estas linguagens são chamadas genericamente de linguagens procedurais (PLs). No caso de uma função escrita em uma linguagem procedural, o servidor de banco de dados não possui nenhum conhecimento interno sobre como interpretar o texto do código fonte da função. Em vez disso, a tarefa é passada para um tratador especial que conhece os detalhes da linguagem. O tratador pode fazer todo o trabalho de análise gramatical, sintática e execução por si mesmo, ou pode servir como um "elo de ligação" entre o PostgreSQL e a implementação existente de uma linguagem de programação. O tratador em si é uma função escrita na linguagem C, compilada como um objeto compartilhado, e carregado conforme necessário, como qualquer outra função escrita na linguagem C (PostgreSQL Global Development Group, 2009).

Segundo o grupo de desenvolvimento global do PostgreSQL (2009), atualmente existem quatro linguagens procedurais disponíveis na distribuição padrão PostgreSQL: PL/pgSQL, PL/Tcl, PL/Perl e PL/Python. Porém, os usuários podem definir outras linguagens através do PostgreSQL.

O PL/pgSQL é a linguagem de procedimentos armazenados mais utilizada no PostgreSQL, devido ser a mais madura e com mais recursos (Mojian, 2001).

2.9 POSTGRESQL 8.4

Segundo o PostgreSQL Global Development Group (2009), o PostgreSQL 8.4 trás 293 modificações entre novos recursos e melhorias em relação a versão 8.3. As mudanças mais numerosas são ferramentas e comandos para administração e monitoramento novos ou melhorados.

Entre as melhorias mais populares estão:

- Restauração Paralela de Bases de Dados, aumentando a velocidade de recuperação de cópia de segurança (*backup*) em até 8 vezes;
- Permissões por Coluna, permitindo um controle mais granular de dados sigilosos;
- Suporte a Configuração Regional por Banco de Dados, tornando o PostgreSQL mais útil em ambientes com múltiplos idiomas;
- *Upgrades In-place* com o `pg_migrator` beta, permitindo migrações da 8.3 para a 8.4 sem tirar o banco do ar por muito tempo;

- Novas Ferramentas de Monitoramento de Consultas, dando aos administradores mais detalhes sobre a atividade das consultas.

2.10 A LINGUAGEM PL/PGSQL

O PL/pgSQL é uma linguagem procedural desenvolvida para o sistema de banco de dados PostgreSQL, tendo como objetivos de projeto da linguagem a criação de uma linguagem procedural carregável que pudesse ser utilizada para criar procedimentos de funções e *triggers*, adicionar estruturas de controle à linguagem SQL, realizar processamentos complexos, herdar todos os tipos de dado, funções e operadores definidos pelo usuário, entre outros (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2009).

A linguagem SQL é a que o PostgreSQL (e a maioria dos bancos de dados relacionais) utiliza como linguagem de comandos. É portátil e fácil de ser aprendida. Entretanto, todas as declarações SQL devem ser executadas individualmente pelo servidor de banco de dados. Isto significa que o aplicativo cliente deve enviar o comando para o servidor de banco de dados, aguardar que seja processado, receber os resultados, realizar algum processamento, e enviar o próximo comando para o servidor. Tudo isto envolve comunicação entre processos e pode, também, envolver tráfego na rede se o cliente não estiver na mesma máquina onde se encontra o servidor de banco de dados (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2009).

Assim como no Oracle e o PL/SQL, a linguagem PL/pgSQL utiliza blocos de processamento e uma série de comandos dentro do servidor de banco de dados, juntando o poder da linguagem procedural com a facilidade de uso da linguagem SQL, e economizando tempo devido não haver sobrecarga de comunicação entre o cliente e o servidor, o que pode aumentar o desempenho consideravelmente (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2009). O quadro 4 mostra a estrutura do bloco PL/pgSQL.

```
[<<label>>]
[DECLARE]
    declarações
BEGIN
    conteúdo 1;
    conteúdo 2;
    conteúdo n;
END;
```

Quadro 4 - Estrutura do bloco PL/pgSQL
Fonte: PostgreSQL Global Development Group, 2009

Também podem ser utilizados na linguagem PL/pgSQL todos os tipos de dados, operadores e funções da linguagem SQL.

2.10.1 Funções

As funções da linguagem PL/pgSQL são correspondentes ao que se chama comumente de *stored procedures* em outras linguagens. O PostgreSQL não possui o conceito *procedure* e quando é necessário a utilização de um código que atenda esse propósito utiliza-se uma função com retorno *void*. (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2009).

O Quadro 5 um exemplo de criação de função no PostgreSQL utilizando PL/pgSQL:

```
CREATE FUNCTION func_escopo() RETURNS integer AS $$
DECLARE
    quantidade integer := 30;
BEGIN
    RAISE NOTICE 'Aqui a quantidade é %', quantidade; -- A
quantidade aqui é 30
    quantidade := 50;
    --
    -- Criar um sub-bloco
    --
    DECLARE
        quantidade integer := 80;
    BEGIN
        RAISE NOTICE 'Aqui a quantidade é %', quantidade; -- A
quantidade aqui é 80
    END;
    RAISE NOTICE 'Aqui a quantidade é %', quantidade; -- A
quantidade aqui é 50
    RETURN quantidade;
END;
$$ LANGUAGE plpgsql;
```

Quadro 5 – Criação de Função no PostgreSQL
Fonte: PostgreSQL Global Development Group, 2009

A função retorna uma variável do tipo inteiro após o encerramento do processamento do bloco. A variável é declarada inicialmente com o valor trinta, logo após a entrada na codificação esse valor é apresentado no console através do comando RAISE NOTICE e depois o seu valor passa a ser cinquenta.

Dentro do bloco existe um sub-bloco com declaração própria onde inclusive ali é declarada uma variável com mesmo nome e tipo da variável declarada no bloco pai, mas com valor diferente, no caso oitenta. Após a exibição da variável do sub-bloco o mesmo é encerrado e o bloco pai segue com seu processamento normal.

Independentemente da variável do sub-bloco ter sido criada após a variável do bloco pai, ela está limitada apenas ao seu bloco, sendo eliminada após o fim do mesmo.

A função se encerra após o a exibição do valor final da variável e seu retorno.

2.11 COMPARAÇÃO ENTRE AS LINGUAGENS PL/SQL E PL/PGSQL

A linguagem PL/pgSQL é semelhante à linguagem PL/SQL em muitos aspectos, sendo consideradas, imperativas, sendo que todas as variáveis devem ser

declaradas. As atribuições, laços e condicionais também são semelhantes (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2009). No entanto existem algumas diferenças entre as duas linguagens:

- No PostgreSQL não existe valor padrão para parâmetros;
- No PostgreSQL os nomes das funções podem ser sobrecarregados. Geralmente isto é utilizado para superar o problema da falta de parâmetros padrão;
- Os cursores não são necessários na linguagem PL/pgSQL, basta adicionar à consulta na instrução FOR;
- No PostgreSQL é necessário utilizar a delimitação por cifrão (\$), ou criar sequências de escape para os apóstrofos presentes no corpo da função;
- Em vez de pacotes, na linguagem PL/pgSQL são utilizados esquemas para organizar as funções em grupos;
- Não existem pacotes nem variáveis no nível de pacotes. Em seu lugar, o estado por sessão pode ser mantido em tabelas temporárias;
- No PostgreSQL não podem ser utilizados nomes de parâmetros idênticos aos das colunas referenciadas na função, mas o Oracle permite que isto seja feito se o nome do parâmetro for qualificado na forma nome_da_função.nome_do_parâmetro.;
- Na linguagem PL/pgSQL não existem procedures, sendo que em seu lugar são utilizadas funções com retorno *void*;

2.12 EXEMPLOS DE DIFERENÇAS:

2.12.1 Conversão de funções

A função demonstrada no Quadro 6 foi escrita em PL/SQL corresponde a uma função simples onde são passados um nome e um sobrenome, sendo que o último opcional. Após a execução do bloco é retornado o valor do nome e o valor do sobrenome intercalados por uma barra (/) se os dois forem passados ou apenas o nome caso nenhum sobrenome seja passado.

```

CREATE OR REPLACE FUNCTION tcc_comparacao1(v_nome VARCHAR,
v_sobrenome VARCHAR)
RETURN varchar IS
BEGIN
    IF v_sobrenome IS NULL THEN
        RETURN v_nome;
    END IF;
RETURN v_nome || '/' || v_sobrenome;
END;
/
show errors;

```

Quadro 6 – Função tcc_comparacao1 escrita em PL/SQL
Fonte: Autoria Própria

Para que uma função semelhante seja escrita para o PostgreSQL em PL/pgSQL alguns pontos devem ser observados:

- A palavra chave RETURN no protótipo da função, não no corpo da função, no PostgreSQL se torna RETURNS. Além disso, IS se torna AS, e é necessário adicionar a cláusula LANGUAGE, porque a linguagem PL/pgSQL não é a única possível.
- No PostgreSQL o corpo da função é considerado como sendo uma literal cadeia de caracteres, portanto é necessário utilizar delimitação por apóstrofos ou cifrão em torno do corpo da função. Isto substitui a barra (/) terminadora da abordagem do Oracle.
- Não existe o comando */show errors* no PostgreSQL, e não há necessidade uma vez que os erros são relatados automaticamente.

No Quadro 7 é apresentado o código da mesma função, escrita em linguagem PL/pgSQL.

```

CREATE OR REPLACE FUNCTION tcc_comparacao1(v_nome varchar,
v_sobrenome
varchar)
RETURNS varchar AS $$
BEGIN
    IF v_sobrenome IS NULL THEN
        RETURN v_nome;
    END IF;
RETURN v_nome || '/' || v_sobrenome;
END;
$$ LANGUAGE plpgsql;

```

Quadro 7 - Função tcc_comparacao1 convertida para PL/pgSQL
Fonte: Autoria Própria

2.12.2 Diferença em recursos nativos

Alguns recursos nativos do PL/SQL não existem na linguagem do PostgreSQL, sendo feitas adaptações. No Quadro 8, o código cria uma *procedure* onde é passado um *e-mail* e então é devolvido o nome do usuário, o domínio do *e-mail* e a sigla do país de origem:

```
create or replace PROCEDURE tcc_comparacao2(v_url IN VARCHAR,
v_usuario out varchar, v_dominio out varchar, v_pais out
varchar)
is
  a_pos1 INTEGER;
  a_pos2 INTEGER;
begin
v_usuario := null;
v_dominio := null;
v_pais := null;
a_pos1 := instr(v_url, '@');
a_pos2 := instr(substr(v_url, a_pos1), '.');
v_usuario := substr(v_url, 1, a_pos1 - 1);
v_dominio := substr(v_url, a_pos1 + 1, a_pos2 - 2);
v_pais := substr(v_url, instr(v_url, '.', -1, 1) + 1);
END;
```

Quadro 8 - Procedure tcc_comparacao2

Fonte: Autoria Própria

É possível notar a utilização do recurso *instr* que é responsável por dizer em qual posição está um caractere dentro uma *string*, sendo esses dois, parâmetros passados pelo usuário.

No PostgreSQL 8.4 essa função não existe nativamente, sendo necessário criar outras funções que desempenhem o mesmo papel.

Como a parametrização da função *instr* é variável já que o usuário pode escolher se quer delimitar a posição inicial ou se ele quer escolher em qual vez o caractere aparece, também é necessário então criar funções que solucionem esse tipo de problema

2.12.3 Exceções

Na linguagem PL/pgSQL, quando uma exceção é capturada pela cláusula *EXCEPTION* todas as alterações no banco de dados desde o começo do bloco (*BEGIN*) são desfeitas automaticamente, ou seja, este comportamento é equivalente ao que seria obtido no Oracle utilizando o código do Quadro 9.


```

BEGIN
    SAVEPOINT s1;
    ... código ...
EXCEPTION
    WHEN ... THEN
        ROLLBACK TO s1;
        ... código ...
    WHEN ... THEN
        ROLLBACK TO s1;
        ... código ...
END;

```

Quadro 9 - Comando para desfazer alterações no Oracle após uma exceção
Fonte: PostgreSQL Global Development Group (2009)

Os códigos de *raise exception* utilizado para registro de exceções também são bastante diferentes:

Oracle:

```

raise_application_error(-20000,
    'Não foi possível criar uma nova tarefa: já há uma
em execução.');
```

PostgreSQL:

```

RAISE EXCEPTION 'Não foi possível criar uma nova tarefa: já há
uma em execução.';
```

Os nomes das exceções suportadas pelo PL/pgSQL são diferentes do Oracle. O conjunto de nomes de exceção nativos é muito maior. Até a versão 8.4 não há como se declarar nomes de exceção definidos pelo usuário no PostgreSQL.

2.12.4 Commit

Não pode ser executada a instrução COMMIT em uma função PL/pgSQL. A função executa dentro de outra transação externa e, portanto, o COMMIT implicaria no término da execução da função.

3 JAVA

É uma linguagem de programação orientada a objetos bastante versátil e atualmente é utilizada nos mais diversos segmentos da indústria. Tem como características principais sua segurança, muitos recursos de rede, já que se comunica com os principais protocolos como o HTTP e o FTP, e grande portabilidade podendo ser executada nos mais diferentes tipos de plataformas. (Oracle Sun Developer Network, 2010).

Foi concebida em 1991 com o nome Project Green na Sun Microsystems. O propósito inicial era o estudo e a análise de interações entre dispositivos eletrônicos e computadores, não exatamente para o desenvolvimento de uma linguagem para aplicativos embarcados. Por ter conceitos vanguardistas o Project Green ficou fadado ao fracasso. Com a popularidade da *internet*, a Sun junto aos seus desenvolvedores receberam a infraestrutura que faltava para dar prosseguimento no seu projeto. Coube somente então adaptar a Oak (como então era conhecida a linguagem na época) para *internet*. Com o advento de novas versões o nome foi alterado para Java (JAVAFREE, 2009).

O Java preza por 5 objetivos básicos (JAVA WIKI, 2011):

- Deve ser orientada a objetos
- Deve permitir a execução do mesmo programa em diferentes tipos de sistemas operacionais.
- Deve possuir suporte nativo a redes de computadores.
- Deve executar códigos de fontes remotas de forma segura.
- Deve ser de fácil utilização selecionando as partes boas de outras linguagens orientadas a objetos.

Java é multiplataforma. Quando um programa Java é compilado um código intermediário é gerado, chamado de *bytecode*. Este *bytecode* é interpretado pelas máquinas virtuais Java (JVMs) para a maioria dos sistemas operacionais. A máquina virtual é a responsável por criar um ambiente multiplataforma, ou seja, se alguém construir um sistema operacional novo, basta criar uma máquina virtual Java que

traduza os *bytecodes* para código nativo. Todas as aplicações Java estarão rodando sem problemas.

Entre outras funções, a máquina virtual Java também é responsável por carregar de forma segura todas as classes do programa, verificar se os *bytecodes* aderem a especificação JVM e se eles não violam a integridade e a segurança do sistema.

A Figura 4 mostra como acontece a compilação e a execução de um programa Java. De um código Java, que está em um arquivo. *Java*, o compilador *javac* gera o *bytecode*:

Um arquivo. *class*. Após isso uma máquina virtual Java executa o *bytecode* e roda o programa.

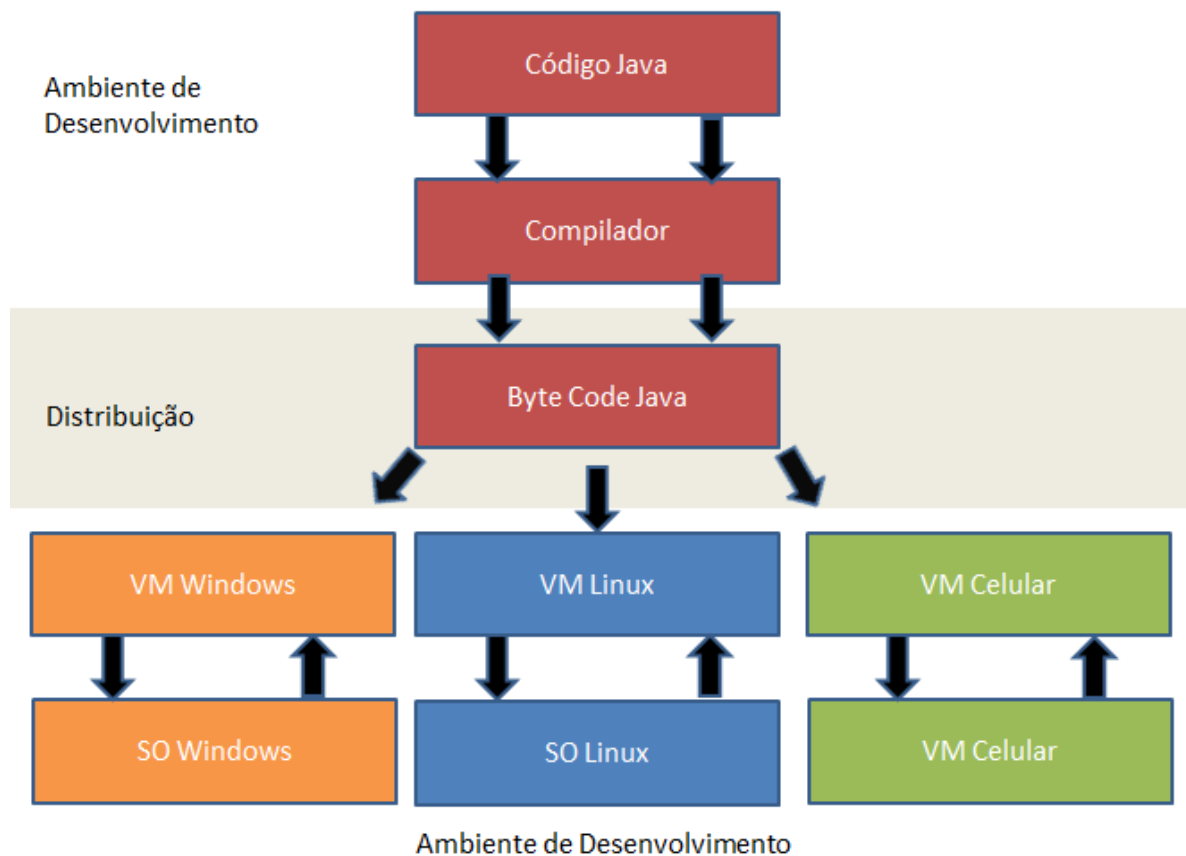


Figura 4 - Funcionamento de uma compilação ou execução de um programa Java
Fonte: Adaptado de JAVA FREE (2009)

Como existe um programa traduzindo um código a cada execução do sistema, poderia-se dizer que Java sempre será mais lenta que as linguagens que geram código nativo do sistema operacional como Delphi, VB ou C + +. Isso era fato até 1996 quando a Sun criou o compilador *Just-in-time* (JIT) que analisa e retira

códigos desnecessários aumentando consideravelmente a velocidade da execução. hoje o Java é mais rápido que o próprio C em muitos aspectos (JAVAFREE, 2009).

A Sun definiu três plataformas voltadas para diferentes ambientes de aplicação e segmentou sua APIs para que elas pertençam a uma delas (JAVAWIKI, 2011).

- *Java Micro Edition* (Java ME) – Voltada para ambientes com recursos limitados.
- *Java Standard Edition* (Java SE) – Voltada para estações de trabalho.
- *Java Enterprise Edition* (Java EE) – Voltada para aplicações corporativas ou em ambiente de internet.

3.1 UTILIZANDO ORACLE E POSTGRESQL EM APLICAÇÕES JAVA

Segundo Doederlein (2009), todos os sistemas gerenciadores de banco de dados relacionais são acessíveis ao Java via JDBC (*Java Data Base Connectivity*) ou via qualquer solução baseada nele, como o JPA (*Java Persistence API*) e o *Hibernate*.

JDBC é o acrônimo de *Java Data Base Connectivity* ou do português Conectividade Java com Banco de Dados e corresponde a um conjunto de especificação de interfaces e classes em Java que padronizam a conectividade com banco de dados. É inspirado no bem sucedido padrão Microsoft de acesso a banco de dados, ODBC (*Open Data Base Connectivity*), porém tem a vantagem de ser multi-plataforma. Além da independência da plataforma, Java também visa obter independência de banco de dados. Isto significa que ao se o SGDB for trocado, espera-se pouca alteração na aplicação (SINTECTUS, 2009).

A Figura 5 representa como pode ser realizada a ligação entre os bancos de dados utilizando a linguagem Java.

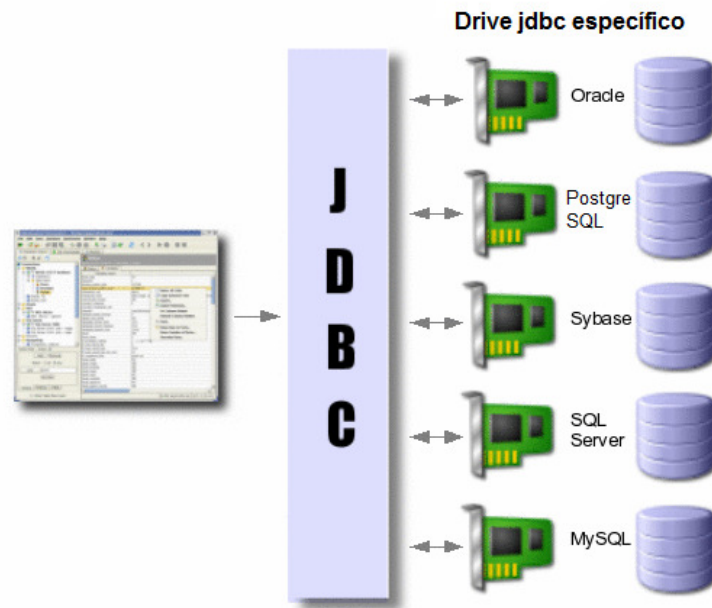


Figura 5 - Estrutura de ligação entre o Java e os Bancos de Dados
Fonte: Minq Software AB 4.2.1 (2005)

Segundo Sintectus (2009), o JDBC permite você utilizar caminhos alternativos para acessar o banco de dados, ou seja, pode-se escolher diferentes *drivers* de diferentes tipos:

- Tipo 1: Conexão através de uma ponte jdbc-odbc. Levando em consideração que a ponte jdbc-odbc já vem incorporada ao JDK, e juntamente com o *driver* ODBC (geralmente fácil de se encontrar), esta opção se torna a mais fácil de se encontrar;
- Tipo 2: O *driver* é obtido a partir de uma API nativa. Isto significa que o *driver* Java faz chamadas nativas a C ou C++ para subrotinas definidas pelo fornecedor do banco de dados. Esta alternativa exige a instalação de *software* cliente;
- Tipo 3: Através de um *driver* específico jdbc. Esta é a melhor opção, uma vez que ganha em desempenho e também é a opção multi-plataforma.

3.1.1 Drivers JDBC Oracle

Segundo Doederlein (2009) existem quatro tipos de *drivers* JDBC que podem ser usados em conexões via JDBC:

- *classes12.jar*: *Driver* compatível com J2SE 1.2 ou superior;
- *ojdbc14.jar*: *Driver* compatível com J2SE 1.4 ou superior;

- *_g.jar (exemplo: *odbc14_g.jar*): Versão de “trace” do *driver*, gera logs muito mais detalhados, sendo excelente para diagnosticar problemas;
- *_dms.jar (exemplo: *odbc14dms.jar*): *Driver* com suporte adicional ao serviço Oracle Dynamic Monitoring System;
- *orai18n.jar*: Arquivo complementar a qualquer um dos *drivers*, fornece suporte adicional a internacionalização.

3.1.2 Drivers JDBC PostgreSQL

De acordo com o PostgreSQL Global Development Group (2009), muitas versões do *driver* JDBC estão disponíveis para PostgreSQL. Isso inclui versões de desenvolvimento, compatibilidade com JDKs mais antigos e versões anteriores do *driver*.

- JDBC 1 para JDK 1.1 - A partir da versão 8.0 do PostgreSQL o suporte ao JDBC foi removido;
- JDBC 2 para JDK 1.2, 1.3;
- JDBC 2 EE para JDK 1.3 + J2EE - Essa versão contém suporte adicional para as classes *javax.sql*;
- JDBC 3 para JDK 1.4, 1.5 - Essa versão contém suporte para SSL e *javax.sql*, mas não necessita o J2EE já que este foi adicionado ao release do J2SE;
- JDBC4 para JDK 1.6 - O suporte para métodos JDBC4 é limitado. Alguns dos métodos novos não existem para esta versão.

No momento da elaboração deste trabalho a versão do JDBC do PostgreSQL se encontra na versão 9, sendo que a não ser que o usuário busque algo específico, como rodar aplicações antigas, o *driver* atual é o recomendado. Suporta o PostgreSQL a partir da versão 7.2 e necessita da JVM versão 1.4 ou superior, sendo esta disponível nas versões JDBC3 e JDBC4, em que recomenda-se a versão JDBC4 para utilizadores da JVM 1.6.

4 ESTUDO DE CASO

Um dos objetivos deste trabalho foi a realização de testes de desempenho em ambas as linguagens. Para a realização dos mesmos foi conveniente a criação de uma aplicação que trabalhasse as linguagens PL/SQL e PLPGSQL, visando a realização de testes para um plano objetivo.

A aplicação desenvolvida objetiva um controle gerencial de uma loja de calçados e foi desenvolvida de uma forma simplificada visando atender os propósitos desse trabalho.

4.1 SOFTWARES UTILIZADOS

Para o desenvolvimento da aplicação foram utilizados os seguintes softwares:

- Eclipse Galileo – *Software* de código livre desenvolvido em Java para construção de programas de computador, principalmente em linguagem Java. Disponível para download em <http://www.eclipse.org/downloads>.
- PgAdmin III 1.10.5. – Ferramenta administrativa para banco de dados PostreSql, inteiramente gratuito. Ele permite escrever SQL simples ou complexas e desenvolver banco de dados com as funções do PostgreSQL. Disponível para download em <http://www.pgadmin.org/download>.
- Oracle SQL Developer versão 2.1.1.6 – Ferramenta gráfica para desenvolvimento de bancos de dados Oracle. Permite navegar em objetos de banco de dados, executar instruções e *scripts* SQL, editar e debugar instruções PL/SQL.

Para elaboração dos diagramas foi utilizado o astah community 6.3 e para a elaboração dos gráficos foi utilizado o Microsoft Excel 2010.

4.2 DESENVOLVIMENTO DA APLICAÇÃO

A aplicação é referente a uma loja de calçados e permite gerenciamento de calçados, clientes, funcionários, ainda permite a realização e alterações de venda.

Todas as tabelas, funções e procedimentos foram criados respeitando sempre as diferenças de sintaxe entre os dois bancos de dados, porém mantendo a lógica básica de cada função ou procedimento.

Na Figura 6, o diagrama ilustra as classes básicas que compõem a estrutura do software e suas relações.

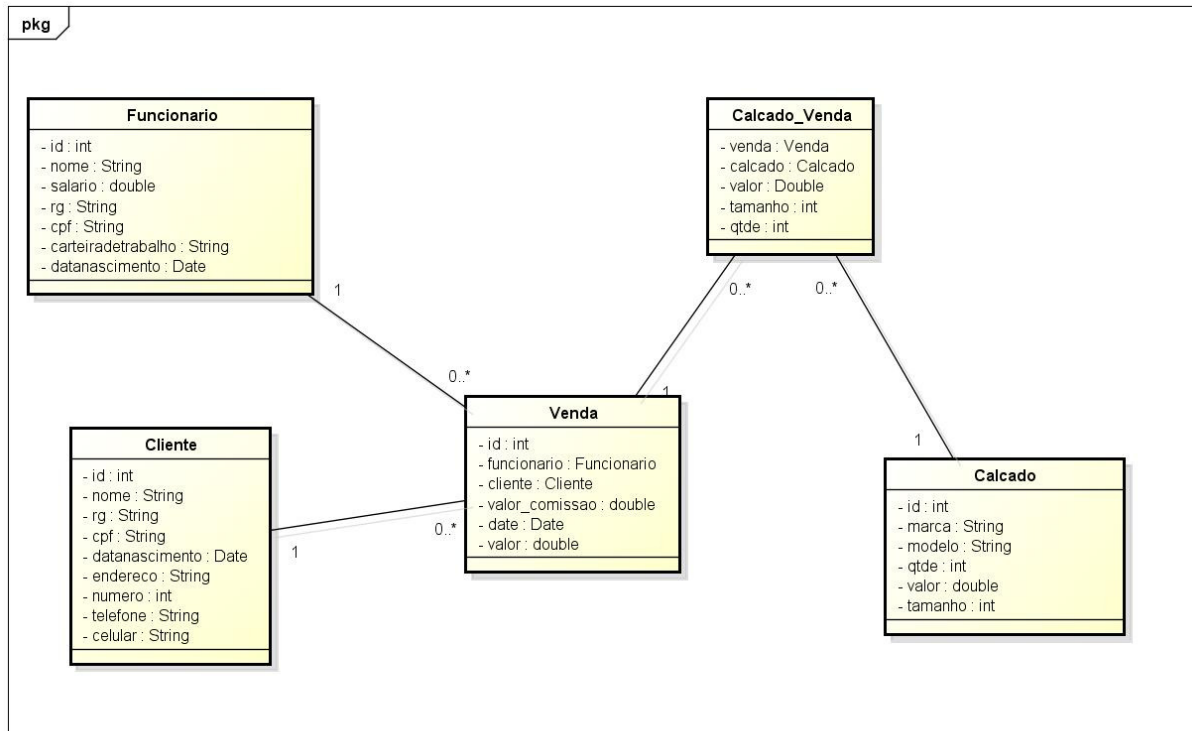


Figura 6 - Diagrama de Classes da Aplicação
Fonte: Autoria Própria

Na etapa referente a criação de tabelas, não houveram diferenças de sintaxe dos comandos SQL, porém é válido ressaltar que a criação de um campo do tipo *Integer* no Oracle faz com que ele seja automaticamente convertido para o tipo *Number(38)*.(SS64, 2011)

4.3 DESENVOLVIMENTO

4.3.1 Criação da estrutura do banco de dados.

Para o desenvolvimento da Loja de Calçados foram utilizados o Banco de Dados Oracle e o PostgreSQL, sendo em cada qual, criadas as tabelas, *procedures* e funções.

Inicialmente foram criadas as tabelas para o armazenamento os dados, sendo definidas as entidades:

- TB_CLIENTE;
- TB_CALÇADO;
- TB_CALÇADO_VENDA;
- TB_FUNCIONARIO;
- TB_VENDA;

Os Quadros 10 e 11 mostram como é feita a criação de uma tabela no Oracle e no PostgreSQL respectivamente.

```

1  create table tb_calçado
2  (
3  cal_id number not null,
4  cal_marca varchar2(30) not null,
5  cal_modelo varchar2(30) not null,
6  cal_qtde number not null,
7  cal_valor double precision not null,
8  cal_tamanho number not null,
9  primary key(cal_id)
10 );

```

Quadro 10 - Código de criação da tabela tb_calçado no banco Oracle
Fonte: Autoria Própria.

```

1  create table tb_calçado
2  (
3  cal_id integer not null,
4  cal_marca varchar(30) not null,
5  cal_modelo varchar(30) not null,
6  cal_qtde integer not null,
7  cal_valor double precision not null,
8  cal_tamanho integer not null,
9  primary key(cal_id)
10 );

```

Quadro 11 - Código de criação da tabela tb_calçado no PostgreSQL
Fonte: Autoria Própria.

Percebe-se a diferença do tipo de dado para inteiros onde no Oracle é o tipo *Number* e no PostgreSQL foi utilizado *Integer*. O banco Oracle possui o tipo de dado *Integer*, porém ele é convertido automaticamente para *Number* com 38 caracteres quando o código de criação de uma tabela é executado.

O Modelo de dados (MER)(Figura 7) descreve as tabelas, seus atributos e seus relacionamentos.

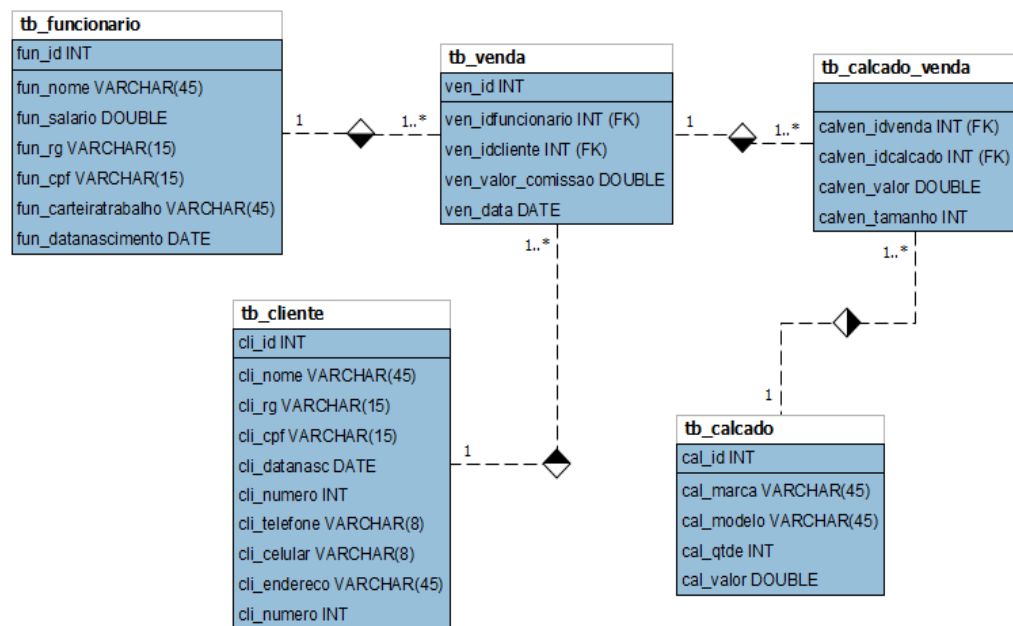


Figura 7 - Diagrama Entidade Relacionamento descrevendo a aplicação
Fonte: Autoria Própria

Na sequência foram criadas as funções FNC_LISTAR. Elas retornam um conjunto de dados para a aplicação através da utilização de cursores³. As funções criadas no sistema foram:

- FNC_LISTAR_CLIENTE;
- FNC_LISTAR_FUNCIONARIO;
- FNC_LISTAR_CALCADO;
- FNC_LISTAR_VENDA;
- FNC_LISTAR_CALCADOPARAVENTA (Essa função é utilizada para listar todos os calçados que possuem quantidade maior que zero para poder disponibilizados para venda.)
- FNC_LISTAR_CALCADOVENDA (Essa função é utilizada para listar todos os calçados que estão associados a uma venda)

Os Quadros 12 e 13 exemplificam como é feita a criação de uma função de listagem no Oracle e no PostgreSQL.

³ Um cursor é o nome de uma parte privada do SQL onde as informações para o processamento de uma declaração específica são mantidas. (Oracle, 2005)

```

1  create or replace function FNC_LISTAR_CALCADO
2  return sys_refcursor
3  as
4  calcado_cursor sys_refcursor;
5  begin
6  open calcado_cursor for
7  'select cal_id, cal_marca, cal_modelo, cal_qtde, cal_valor, cal_tamanho
8  from tb_calcado order by cal_id';
9  return calcado_cursor;
10 commit;
11 end;

```

Quadro 12 - Código de criação da função fnc_listar_calcado no Oracle
Fonte: Autoria Própria.

```

1  CREATE OR REPLACE FUNCTION fnc_listar_calcado()
2  RETURNS refcursor AS
3  $$
4  DECLARE
5  calcado_cursor refcursor;
6  BEGIN
7  OPEN calcado_cursor for select cal_id, cal_marca,
8  cal_modelo, cal_qtde, cal_valor, cal_tamanho
9  from tb_calcado order by cal_id;
10 RETURN calcado_cursor;
11 END;
12 $$
13 language plpgsql

```

Quadro 13 - Código de criação da função fnc_listar_calcado no Postgresql
Fonte: Autoria Própria.

O passo seguinte foi o de criação das *procedures* OBTER_PK, que retornam o último id registrado no banco. Usado na *procedure* de atualização (PRC_ATUALIZAR) no processo de inserção de registros para a geração automática de id.

- PRC_OBTER_PK_CLIENTE;
- PRC_OBTER_PK_FUNCIONARIO;
- PRC_OBTER_PK_CALÇADO;
- PRC_OBTER_PK_VENDA;

Os quadros 14 e 15 mostram como é a estrutura de uma *procedure* OBTER_PK.

```

1  create or replace procedure prc_obter_pk_calcado
2  (pk out tb_calcado.cal_id%type)
3  is
4  begin
5      select max(cal_id) into pk from tb_calcado;
6  if pk IS null then
7      pk := 0;
8  end if;
9  commit;
10 end;

```

Quadro 14 - Código de criação da *procedure* *prc_obter_pk_calcado* no Oracle
Fonte: Autoria Própria.

```

1  create or replace function prc_obter_pk_calcado
2  (pk out tb_calcado.cal_id%type)
3  returns integer as
4  $$
5  begin
6      select max(cal_id) into pk from tb_calcado;
7  if pk IS null then
8      pk := 0;
9  end if;
10 end;
11 $$
12 language plpgsql;

```

Quadro 15 - Código de criação da *procedure* *prc_obter_pk_calcado* no PostgreSQL
Fonte: Autoria Própria.

As *procedures* PRC_ATUALIZAR são responsáveis pela inserção ou alteração dos dados. Quando estas *procedures* são chamadas, é verificado qual o id passado junto para poder definir o propósito da função para aquela execução. Se o id passado for nulo significa que é uma inserção, então através das *procedures* PRC_OBTOR_PK é verificado qual o id cadastrado na última inserção e definido qual o id do novo registro. Caso seja passado algum valor como parâmetro no chamamento da função, é então definido que o registro que possui esse valor no seu campo id será atualizado.

- PRC_ATUALIZAR_CLIENTE;
- PRC_ATUALIZAR_FUNCIONARIO;
- PRC_ATUALIZAR_CALCADO;
- PRC_ATUALIZAR_VENDA;

Nos Quadros 16 e 17 seguem os códigos de criação da PRC_ATUALIZAR_CALCADO para o Oracle e para o PostgreSQL.

É possível notar como existe pouca diferença estrutural das *procedures* nas duas linguagens.

```
1  create or replace procedure prc_atualizar_calçado
2  (
3      tmp_cal_id IN tb_calçado.cal_id%TYPE,
4      tmp_cal_marca IN tb_calçado.cal_marca%TYPE,
5      tmp_cal_modelo IN tb_calçado.cal_modelo%TYPE,
6      tmp_cal_qtde IN tb_calçado.cal_qtde%TYPE,
7      tmp_cal_valor IN tb_calçado.cal_valor%TYPE,
8      tmp_cal_tamanho IN tb_calçado.cal_tamanho%TYPE
9  ) IS
10     var_aux_ultimo_id NUMBER;
11     begin
12         if tmp_cal_id = 0 then
13             prc_obter_pk_calçado(var_aux_ultimo_id);
14             var_aux_ultimo_id := var_aux_ultimo_id + 1;
15         end if;
16         if tmp_cal_id = 0 then
17             insert into tb_calçado values (var_aux_ultimo_id, tmp_cal_marca,
18             tmp_cal_modelo, tmp_cal_qtde, tmp_cal_valor, tmp_cal_tamanho);
19         else
20             update tb_calçado set cal_marca = tmp_cal_marca,
21             cal_modelo = tmp_cal_modelo,
22             cal_qtde = tmp_cal_qtde,
23             cal_valor = tmp_cal_valor,
24             cal_tamanho = tmp_cal_tamanho
25             where cal_id = tmp_cal_id;
26         end if;
27         commit;
28     end;
```

Quadro 16 - Código de criação da prc_atualizar_calçado no Oracle
Fonte: Autoria Própria.

```

1  create or replace function prc_atualizar_calçado
2  (
3      tmp_cal_id IN tb_calçado.cal_id%TYPE,
4      tmp_cal_marca IN tb_calçado.cal_marca%TYPE,
5      tmp_cal_modelo IN tb_calçado.cal_modelo%TYPE,
6      tmp_cal_qtde IN tb_calçado.cal_qtde%TYPE,
7      tmp_cal_valor IN tb_calçado.cal_valor%TYPE,
8      tmp_cal_tamanho IN tb_calçado.cal_tamanho%TYPE
9  ) returns void AS
10 $$
11 declare
12     var_aux_ultimo_id INTEGER;
13 begin
14     if tmp_cal_id = 0 then
15         var_aux_ultimo_id := prc_obter_pk_calçado();
16         var_aux_ultimo_id := var_aux_ultimo_id + 1;
17     end if;
18     if tmp_cal_id = 0 then
19         insert into tb_calçado values (var_aux_ultimo_id, tmp_cal_marca,
20             tmp_cal_modelo, tmp_cal_qtde, tmp_cal_valor, tmp_cal_tamanho);
21     else
22         update tb_calçado set cal_marca = tmp_cal_marca,
23             cal_modelo = tmp_cal_modelo,
24             cal_qtde = tmp_cal_qtde,
25             cal_valor = tmp_cal_valor,
26             cal_tamanho = tmp_cal_tamanho
27         where cal_id = tmp_cal_id;
28     end if;
29 end;
30 $$
31 language plpgsql;

```

Quadro 17 - Código de criação da *procedure* *prc_atualizar_calçado* no PostgreSQL
Fonte: Autoria Própria.

As *procedures* PRC_OBTER_POR_ID são responsáveis por trazer um registro baseado em um id passado por parâmetro. É passado o valor do id como dado de entrada (IN) e os dados referentes a esse registro são retornados como parâmetros de saída (OUT).

- PRC_OBTER_POR_ID_CLIENTE;
- PRC_OBTER_POR_ID_FUNCIONARIO;
- PRC_OBTER_POR_ID_CALCADO.
- PRC_OBTER_POR_ID_VENDA.

Os Quadros 18 e 19 exemplificam como é feita uma *procedure* *obter_por_id*.

```

1  create or replace procedure prc_obter_por_id_calçado
2  (
3    tmp_cal_id IN tb_calçado.cal_id%TYPE,
4    tmp_cal_marca OUT tb_calçado.cal_marca%TYPE,
5    tmp_cal_modelo OUT tb_calçado.cal_modelo%TYPE,
6    tmp_cal_qtde OUT tb_calçado.cal_qtde%TYPE,
7    tmp_cal_valor OUT tb_calçado.cal_valor%TYPE,
8    tmp_cal_tamanho OUT tb_calçado.cal_tamanho%TYPE
9  )
10 IS
11 BEGIN
12     SELECT
13     cal_marca, cal_modelo, cal_qtde, cal_valor, cal_tamanho INTO
14     tmp_cal_marca, tmp_cal_modelo, tmp_cal_qtde, tmp_cal_valor, tmp_cal_tamanho
15     from tb_calçado where cal_id = tmp_cal_id;
16     commit;
17 end;

```

Quadro 18 - Código de criação da *procedure* *prc_obter_por_id_calçado* no Oracle

```

1  create or replace function prc_obter_por_id_calçado
2  (
3    tmp_cal_id IN tb_calçado.cal_id%TYPE,
4    tmp_cal_marca OUT tb_calçado.cal_marca%TYPE,
5    tmp_cal_modelo OUT tb_calçado.cal_modelo%TYPE,
6    tmp_cal_qtde OUT tb_calçado.cal_qtde%TYPE,
7    tmp_cal_valor OUT tb_calçado.cal_valor%TYPE,
8    tmp_cal_tamanho OUT tb_calçado.cal_tamanho%TYPE
9  )
10 $$
11 BEGIN
12     SELECT
13     cal_marca, cal_modelo, cal_qtde, cal_valor, cal_tamanho INTO
14     tmp_cal_marca, tmp_cal_modelo, tmp_cal_qtde, tmp_cal_valor, tmp_cal_tamanho
15     from tb_calçado where cal_id = tmp_cal_id;
16 end;
17 $$
18 language plpgsql;

```

Quadro 19 - Código de criação da *procedure* *prc_obter_por_id_calçado* no PostgreSQL
Fonte: Autoria Própria.

As *procedures* PRC_APAGAR servem para eliminar uma tupla de uma tabela através do id passado.

- PRC_APAGAR_CLIENTE;
- PRC_APAGAR_FUNCIONARIO;
- PRC_APAGAR_CALCADO.
- PRC_APAGAR_CALCADO_VENDA.

```

1  create or replace procedure prc_apaga_calçado
2  (
3  var_cal_id number
4  )
5  is
6  begin
7  delete from tb_calçado where cal_id = var_cal_id;
8  end;

```

Quadro 20 - Código de criação da *procedure* *prc_apaga_calçado* no Oracle
Fonte: Autoria Própria.

```

1  create or replace function prc_apaga_calçado
2  (
3  var_cal_id integer
4  )
5  returns void AS
6  $$
7  begin
8  delete from tb_calçado where cal_id = var_cal_id;
9  end;
10 $$
11 language plpgsql;

```

Quadro 21 - Código de criação da *procedure* *prc_apaga_calçado* no PostgreSQL
Fonte: Autoria Própria.

A última parte relativa a criação de funções e *procedures* foi na criação de funções relativas a atualização de venda.

- PRC_ALTERAR_QTDE_CALCADO (Procedure responsável por atualizar a quantidade de calçados disponíveis na tabela *tb_calçados*)
- PRC_ATUALIZA_CALCADO_VENDA (Procedure responsável por atualizar o número de calçados envolvidos em uma venda).

4.3.2 Desenvolvimento da aplicação através do Java.

A estrutura da aplicação foi criada em cima de três pacotes principais:

O pacote domínio traz todos os pontos básicos estruturais. Ele se divide nos seguintes sub-pacotes:

- Dao – Contém as classes responsáveis pelas regras de acesso ao banco de dados. Todas as execuções das funções e procedimentos são realizadas pelas classes deste pacote.
- Pojo – Contem as classes de modelo que representam as informações de uma entidade.

O Quadro 22 mostra o código de uma classe pojo.


```

1 package dominio.pojos;
2
3 public class Calcado {
4
5     private int id;
6     private String marca;
7     private String modelo;
8     private int qtde;
9     private Double valor;
10    private int Tamanho;
11
12    public int getId() {return id;}
13    public void setId(int id) {this.id = id;}
14    public String getMarca() {return marca;}
15    public void setMarca(String marca) {this.marca = marca;}
16    public String getModelo() {return modelo;}
17    public void setModelo(String modelo) {this.modelo = modelo;}
18    public int getQtde() {return qtde;}
19    public void setQtde(int qtde) {this.qtde = qtde;}
20    public Double getValor() {return valor;}
21    public void setValor(Double valor) {this.valor = valor;}
22    public int getTamanho() {return Tamanho;}
23    public void setTamanho(int tamanho) {Tamanho = tamanho;}
24    public String toString() {return modelo;}
25 }

```

Quadro 22 - Código da classe Calcado
Fonte: Autoria Própria.

Um POJO representa uma entidade da aplicação e estabelece uma correspondência com o banco de dados onde os atributos da classe tem um relacionamento um-a-um com os campos da tabela.(DEMOISELLE, 2010)

Um POJO é uma classe que consta essencialmente de atributos e métodos acessores e modificadores desses atributos. (DEMOISELLE, 2010)

- Tabela – Contém a classe responsável por conter a classe que serve como modelo para as tabelas que são utilizadas em buscas dentro da aplicação.

O segundo pacote principal, infraestrutura, contém as classes de conexão com o banco de dados.

E o terceiro e último pacote, visão, contém as classes de interface do usuário onde este faz utilização da aplicação.

A aplicação ainda conta com a utilização de JARs, que são conjuntos de classes, responsáveis por trazerem recursos importantes a mesma.

Os seguintes JARs foram utilizados:

- classes12 – Responsável por fazer a conexão com o banco de dados Oracle.

- postgresql-9.0-801.jdbc3.jar – Responsável por fazer a conexão com o banco de dados PostgreSQL
- p6spy.jar e sqlprofiler.jar – São os JARs responsáveis pela realização dos testes de desempenho.
- forms-1.3.0 – Jar utilizado na criação das telas.

A Figura 8 ilustra como ficou a estrutura de pacotes dentro da IDE Eclipse.

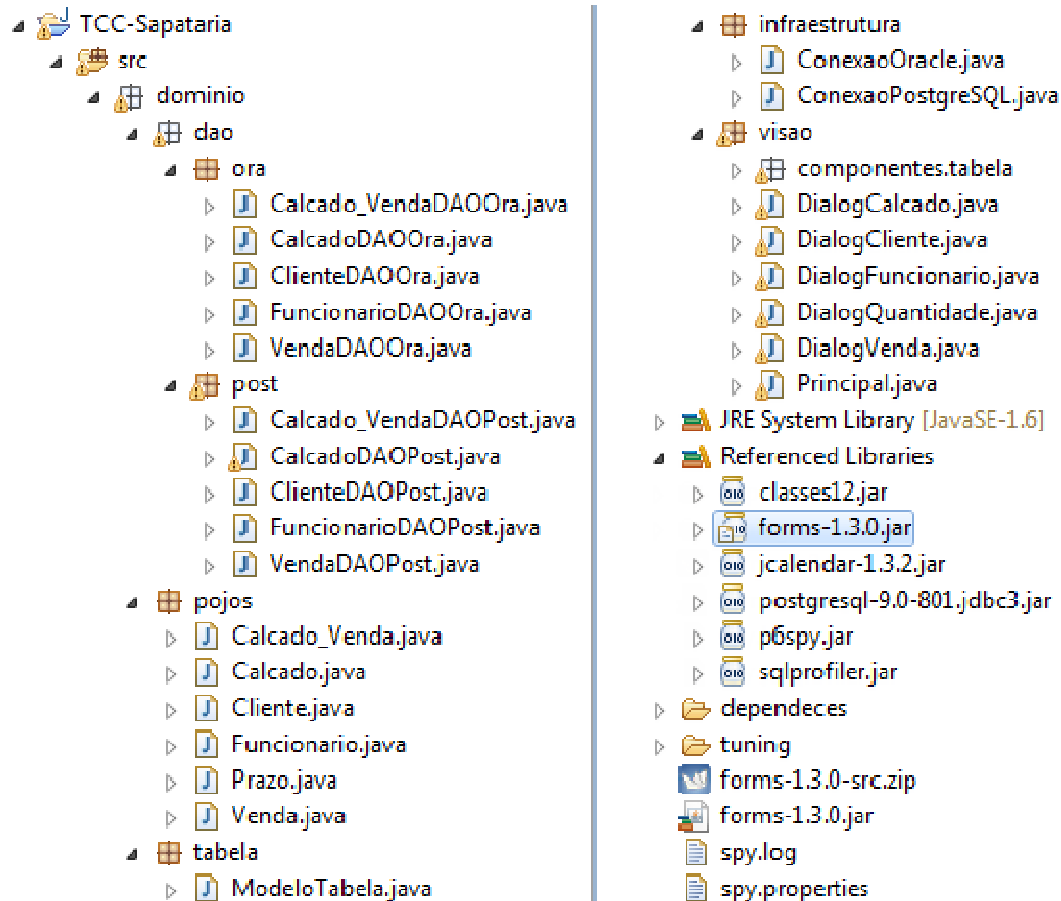


Figura 8 - Estrutura de pacotes da aplicação
Fonte: Autoria Própria

Para a conexão com o banco de dados foram criadas duas classes semelhantes, a `ConexaoOracle.java` responsável pela conexão com o Oracle e a `ConexaoPostgreSQL.java`. (Quadro 23 E Quadro 24).

```

1 package infraestrutura;
2
3 import java.sql.Connection;
4
5
6
7 public class ConexaoOracle {
8     private static Connection conexao;
9     public static Connection getConexao()
10    {
11        try{
12            Class.forName("com.p6spy.engine.spy.P6SpyDriver");
13            conexao = DriverManager.getConnection(
14                "jdbc:oracle:thin:@localhost:1521:xe", "system","admin");
15        } catch (SQLException e){
16            e.printStackTrace();
17        } catch (ClassNotFoundException e) {
18            e.printStackTrace();
19        }
20        return conexao;
21    }
22 }

```

Quadro 23 - Classe conexão com o Oracle
Fonte: Autoria Própria.

```

1 package infraestrutura;
2
3 import java.sql.Connection;
4
5
6
7 public class ConexaoPostgreSQL {
8     private static final String url = "jdbc:postgresql://localhost:5432/TCC";
9     private static Connection conexao;
10
11    public static Connection getConexao()
12    {
13        try{
14            Class.forName("com.p6spy.engine.spy.P6SpyDriver");
15            conexao = DriverManager.getConnection(url, "postgres", "admin");
16        } catch (ClassNotFoundException e) {
17            e.printStackTrace();
18        } catch (SQLException e){
19            e.printStackTrace();
20        }
21
22        return conexao;
23    }
24 }

```

Quadro 24 - Classe de conexão com PostgreSQL
Fonte: Autoria Própria.

É possível notar o trecho:

“Class.forName(“com.p6spy.engine.spy.P6SpyDriver”)”.

Neste local seriam descritos os *drivers* de conexão com os bancos, no caso `org.postgresql.Driver` para o PostgreSQL e `oracle.jdbc.driver.OracleDriver` para o Oracle, porém eles foram substituídos pelo *driver* de conexão com a ferramenta responsável pelos teste de desempenho, o PS6Spy.

```
53#driver do postgresql
54realdriver=org.postgresql.Driver
55
56#driver do oracle
57realdriver2=oracle.jdbc.driver.OracleDriver
```

Quadro 25 - Trecho do código do arquivo de propriedades spy.log
Fonte: Autoria Própria.

A ferramenta de testes possui um arquivo de propriedades, `spy.properties`. Todas as conexões feitas aos bancos de dados são direcionadas ao PS6Spy, que por sua vez redireciona as conexões baseadas nesse arquivo de propriedades. (Quadro 25)

A Figura 9 mostra a tela principal da aplicação.

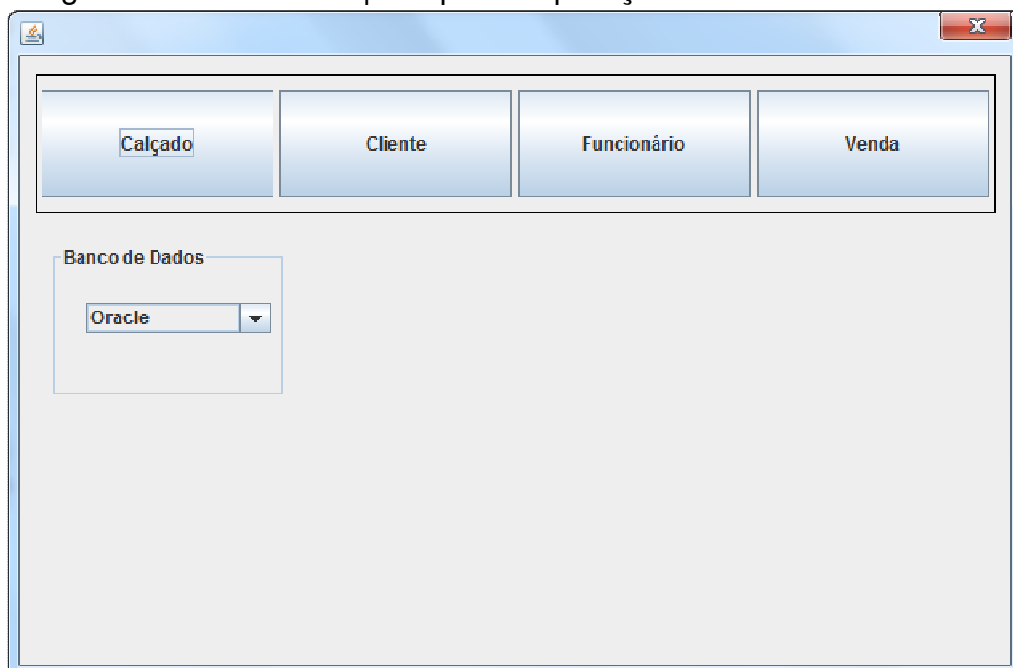


Figura 9 - Tela principal da aplicação
Fonte: Autoria Própria

São quatro botões que abrem as telas referentes as classes que compõem a aplicação. Com exceção da tela de venda, todas as outras seguem o mesmo padrão.

No *comboBox* dentro do painel “Banco de Dados” é possível escolher qual banco será utilizado na tela que será aberta. Ele associa cada item do *comboBox* a

um número inteiro, que é passado como parâmetro no chamamento de uma tela. Esse número depois é utilizado para decidir qual banco de dados foi escolhido toda vez que for preciso fazer um chamamento de função ou *procedure*.

A Figura 10 mostra como o usuário realiza a inserção de um calçado novo no banco de dados.

Primeiro o usuário clica em novo, após o preenchimento dos dados clica em “gravar”. O id do novo registro é gerado automaticamente através da *procedure* de inserção.

A interface de usuário para a inserção de calçados é exibida em uma janela com o título "Calçado". No topo, há uma barra de ferramentas com os botões "Novo", "Gravar", "Cancelar", "Excluir" e "Consultar". Abaixo, há uma seção "Dados" com campos de entrada para "Código", "Marca", "Modelo", "Quantidade", "Valor" e "Tamanho". Os campos "Marca" e "Modelo" contêm os valores "CONVERSE" e "ALL-STAR", respectivamente. Os campos "Quantidade", "Valor" e "Tamanho" contêm os valores "30", "100" e "40", respectivamente. Abaixo da seção "Dados", há uma seção "Calçados" com uma lista vazia.

Figura 10 - Inserção de Calçado
Fonte: Autoria Própria

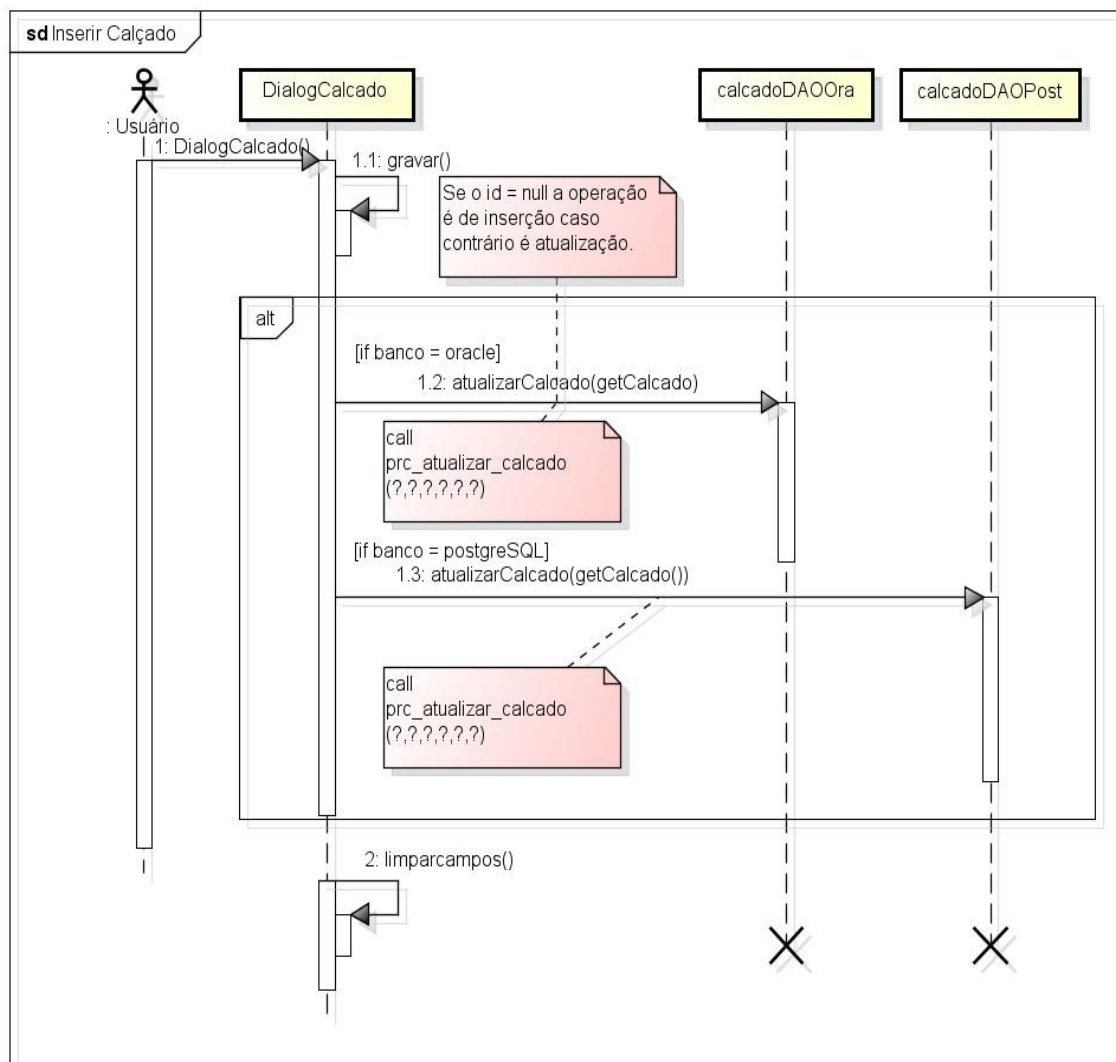
Como a gravação e a alteração dos dados são realizadas através da mesma *procedure*, a definição de qual instrução será executada depende se os dados enviados para a *procedure* possuem um id.

Quando um dado é recuperado através da consulta conforme na Figura 11, o seu id é passado para o seu campo respectivo na tela no momento que o usuário clica no registro para poder alterar os dados (Figura 12). O fato de existir um id escrito no campo "Id" da tela, garante que está sendo feita uma alteração referente ao registro que possui esse valor como chave. Caso o usuário deseje inserir um novo item de calçado, é necessário que ele clique em "Cancelar" para que os campos sejam limpos e a associação seja desfeita.

Figura 11 - Tela de calçados no momento de uma consulta
Fonte: Autoria Própria

Figura 12 - Tela de calçado no momento de uma alteração
Fonte: Autoria Própria

A Figura 13 mostra quais são os passos para que seja feita uma gravação ou uma atualização dos dados.



powered by astah®

Figura 13 - Diagrama de seqüência para a inserção de calçados
Fonte: Autoria Própria

Quando o usuário clica em gravar, é verificado qual é o banco de dados escolhido, para que seja definido qual método será executado pela aplicação. Após essa definição é visto se o valor do id será nulo ou não, definindo também qual o comportamento da *procedure*. Então o método `atualizarCalçado()` é executado tendo como parâmetro os dados escritos na tela.

O Quadro 26 mostra o código da função `atualizarCalçado()` na versão Oracle. Essa função está localizada dentro da classe `CalcadoDAOOracle`, que é a classe que contém os métodos de acesso a dados do Oracle. O processo para o `postgreSQL` está localizado na classe `CalcadoDAOPostgreSQL` e é basicamente o mesmo mudando apenas qual a classe de conexão será chamada.

```

66 public void atualizarCalcado(Calcado calcado) throws SQLException {
67     Connection conexao = null;
68     CallableStatement procedimento = null;
69     try {
70         conexao = ConexaoOracle.getConexao();
71         procedimento = conexao.prepareCall(
72             "{call prc_atualizar_calcado(?,?,?,?,?,?)}");
73         procedimento.setInt(1, calcado.getId());
74         procedimento.setString(2, calcado.getMarca());
75         procedimento.setString(3, calcado.getModelo());
76         procedimento.setInt(4, calcado.getQtde());
77         procedimento.setDouble(5, calcado.getValor());
78         procedimento.setInt(6, calcado.getTamanho());
79         procedimento.execute();
80     } catch(Exception ex) {
81         ex.printStackTrace();
82     } finally {
83         procedimento.close();
84         conexao.close();
85     }
86 }

```

Quadro 26 - Código referente ao método atualizar calçado na classe CalcadoDAOOracle
Fonte: Autoria Própria.

Todas as lógicas de execução de *procedures* ou funções seguem essa lógica básica de verificação de qual banco foi escolhido e qual classe DAO deverá ser ter seu método executado.

O objeto `CallableStatement` é responsável por possibilitar a realização de chamadas de *procedures* ou funções. Com ele é possível realizar chamadas com parâmetros de entrada (IN), de saída (OUT) e ambos (IN OUT).

O código para o chamamento da *procedure* `prc_atualizar_calcado` ficou da seguinte forma:

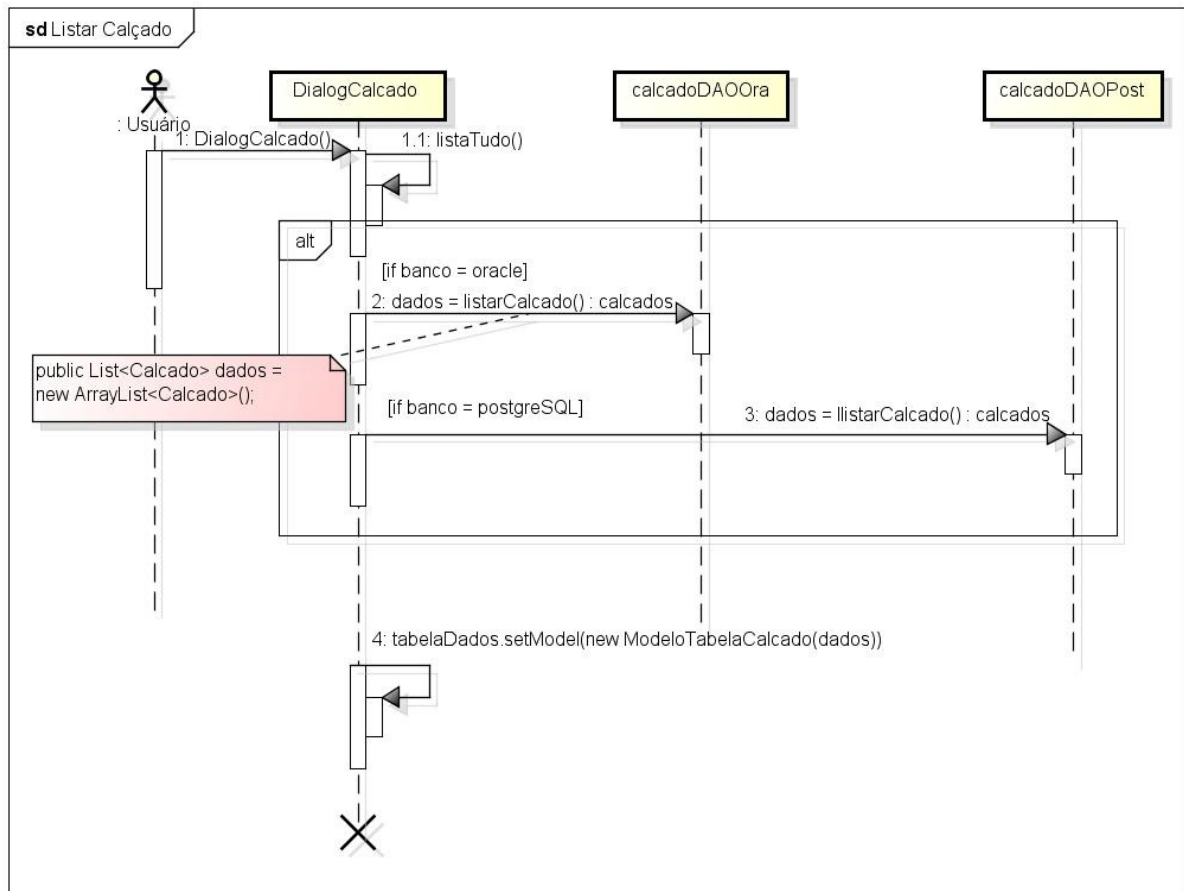
```
call prc_atualizar_calcado(?,?,?,?,?,?)
```

A *procedure* é do tipo IN, onde os pontos de interrogação são os parâmetros que são passados como entrada.

Através do método `Statement` e do método `PreparedStatement`, o objeto `CallableStatement` trata das instruções SQL. Esses objetos são criados através do método de conexão `prepareCall`. O código da sintaxe de criação de um objeto `CallableStatement` que realiza a chamada a chamada para a *procedure* `prc_atualizar_calcado` é o seguinte:

```
CallableStatement procedimento = conexao.prepareCall(
    "{call prc_atualizar_calcado(?,?,?,?,?,?)}");
```


O diagrama de sequência (Figura 14) mostra como é feita uma consulta na aplicação.



powered by astah®

Figura 14 - Diagrama de Sequência para listagens de calçado
Fonte: Autoria Própria

Como dito, as consultas também seguem a lógica básica de verificação de qual banco será utilizado para decidir qual classe DAO será chamada.

Os Quadros 27 e 28 exibem o código do método responsável pela listagem de calçados de cada linguagem. É possível notar a leve diferença na *string* utilizada para chamar a função no banco.

```

31 public List<Calcado> listarCalcado() throws Exception {
32     Connection conexao = null;
33     CallableStatement procedimento = null;
34     List<Calcado> calcados = new ArrayList<Calcado>();
35
36     try {
37         conexao = ConexaoOracle.getConexao();
38         procedimento = conexao.prepareCall(
39             "{call ? := fnc_listar_calcado()}");
40         procedimento.registerOutParameter(1, OracleTypes.CURSOR)
41         procedimento.execute();
42         ResultSet set = (ResultSet) procedimento.getObject(1);
43         while(set.next()) {
44             Calcado cal = new Calcado();
45             cal.setId(set.getInt(1));
46             cal.setMarca(set.getString(2));
47             cal.setModelo(set.getString(3));
48             cal.setQtde(set.getInt(4));
49             cal.setValor(set.getDouble(5));
50             cal.setTamanho(set.getInt(6));
51             calcados.add(cal);
52         }
53
54     } catch(Exception ex) {
55         ex.printStackTrace();
56     } finally {
57         procedimento.close();
58         conexao.close();
59     }
60     return calcados;
61 }

```

Quadro 27 - Método listarCalcado na classe CalcadoDAOOracle
Fonte: Autoria Própria.

No Oracle, quando uma função ou *procedure* retorna um parâmetro é utilizado um ponto de interrogação recebendo o retorno da função ou *procedure*.

```
call ? := fnc_listar_calcado()
```

```

34 public List<Calcado> listarCalcado() throws Exception {
35     Connection conexao = null;
36     List<Calcado> calcados = new ArrayList<Calcado>();
37     try {
38         conexao = ConexaoPostgreSQL.getConexao();
39         CallableStatement proc = conexao.prepareCall(
40             "{? = call fnc_listar_calcado()}");
41         proc.registerOutParameter(1, Types.OTHER);
42         proc.execute();
43         ResultSet set = (ResultSet) proc.getObject(1);
44         while (set.next()) {
45             Calcado cal = new Calcado();
46             cal.setId(set.getInt(1));
47             cal.setMarca(set.getString(2));
48             cal.setModelo(set.getString(3));
49             cal.setQtde(set.getInt(4));
50             cal.setValor(set.getDouble(5));
51             cal.setTamanho(set.getInt(6));
52             calcados.add(cal);
53         }
54         set.close();
55         proc.close();
56     } catch (Exception ex) {
57         ex.printStackTrace();
58     } finally {
59         conexao.close();
60     }
61     return calcados;
62 }

```

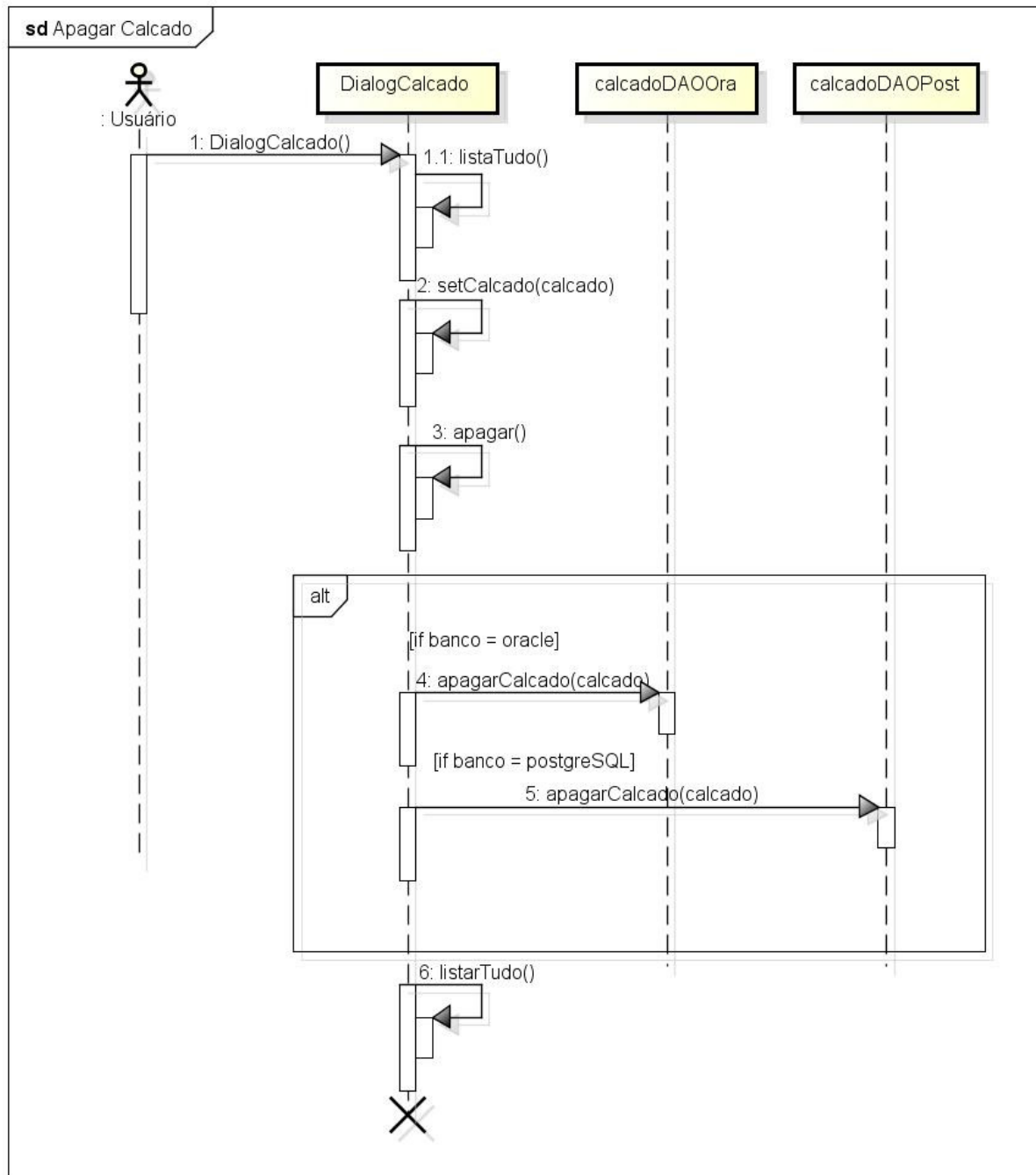
Quadro 28 - Método listarCalcado() na classe CalcadoDAOPostgreSQL
Fonte: Autoria Própria.

No postgresQL a ideia é basicamente a mesma porém o ponto de interrogação muda de lugar.

```
? = call fnc_listar_calcado
```

O diagrama de sequencia a da Figura 15 exemplifica como é realizada uma exclusão de registro na base de dados.

Primeiro o usuário lista todos os registros da tabela, escolhe qual deles será excluído e depois realiza a exclusão. Novamente a aplicação verifica qual banco de dados foi escolhido e baseado nisso chama o método da classe DAO responsável pela exclusão.



powered by astah®

Figura 15 - Diagrama de Sequência relativo a uma exclusão de calçado

Fonte: Autoria Própria

O Quadro 29 exhibe como é o código dos métodos responsáveis pela a exclusão.

```

85 public static void apagaCalcado(Calcado calcado) throws SQLException{
86     Connection conexao = null;
87     CallableStatement procedimento = null;
88     try {
89         conexao = ConexaoOracle.getConexao();
90         procedimento = conexao.prepareCall(
91             "{call prc_apagar_calcado(?)}");
92         procedimento.setInt(1, calcado.getId());
93         procedimento.execute();
94     } catch(Exception ex) {
95         ex.printStackTrace();
96     } finally {
97         procedimento.close();
98         conexao.close();
99     }
.00 }

```

Quadro 29 - Código do método apagaCalcado na classe CalcadoDAOOracle
Fonte: Autoria Própria

O método na versão PostgreSQL é praticamente o mesmo, mudando apenas a linha 89 onde é substituída a classe que chama o método getConexao().

A Figura 16 exibe a tela de venda. Nela o usuário pode realizar uma venda nova ou atualizar uma já realizada.

Figura 16 - Tela de Venda
Fonte: Autoria Própria

O primeiro passo para realizar uma venda é adicionar quais calçados serão utilizados. O usuário clica no botão "+" e escolhe qual calçado será adicionado à venda (Figura 17).

Parâmetros

Calçados

Id	Marca	Modelo	Valor	Tamanho	Quantidade
1	ADIDAS	SUPERSTAR	200	40	10
2	NIKE	AIR MAX 360	300	41	10
3	CONVERSE	ALLSTAR	90	39	10
4	KICHUTE	VAGAL	30	42	10
5	CONVERSE	ALL-STAR	100	40	30

Valor Comissao: R\$ 0.0

Vender! Buscar Ve... Atualizar! Cancelar

Dados

Lódigo

Funcionário

Cliente

Data: 05/07/2011

Figura 17 - Tela de adição de calçados na venda

Fonte: Autoria Própria

Após a escolha do calçado o usuário tem a possibilidade de definir qual será a quantidade (Figura 18):

Informe a quantidade

1

+

-

OK

Id	Marca	Modelo	Valor	Tamanho	Quantidade
1	ADIDAS	SUPERSTAR	200	40	10
2	NIKE	AIR MAX 360	300	41	10
3	CONVERSE	ALLSTAR	90	39	10
4	KICHUTE	VAGAL	30	42	10
5	CONVERSE	ALL-STAR	100	40	30

Figura 18 - Tela de adição de calçados na venda no momento que a quantidade é informada.

Fonte: Autoria Própria

Com isso o valor e o da comissão são automaticamente calculados. O usuário também precisa definir qual foi o funcionário que realizou a venda e qual o cliente comprador, conforme exibido na Figura 19.

Id	Marca	Modelo	Valor	Tamanho	Quantidade
5	CONVERSE	ALL-STAR	100	40	1

Valor Total: R\$ 100.0

Valor Comissão: R\$ 13.0

Botões: Vender!, Buscar Ve..., Atualizar!, Cancelar, Novo

Dados:

Código: []

Funcionário: A [+]

Cliente: JOSÉ CLIENTE [+]

Data: 06/07/2011 []

Figura 19 - Tela de venda com todos os dados inseridos prestes a ser realizada

Fonte: Autoria Própria

Após isso o usuário clica no botão “Vender!” e o processo de gravação é iniciado.

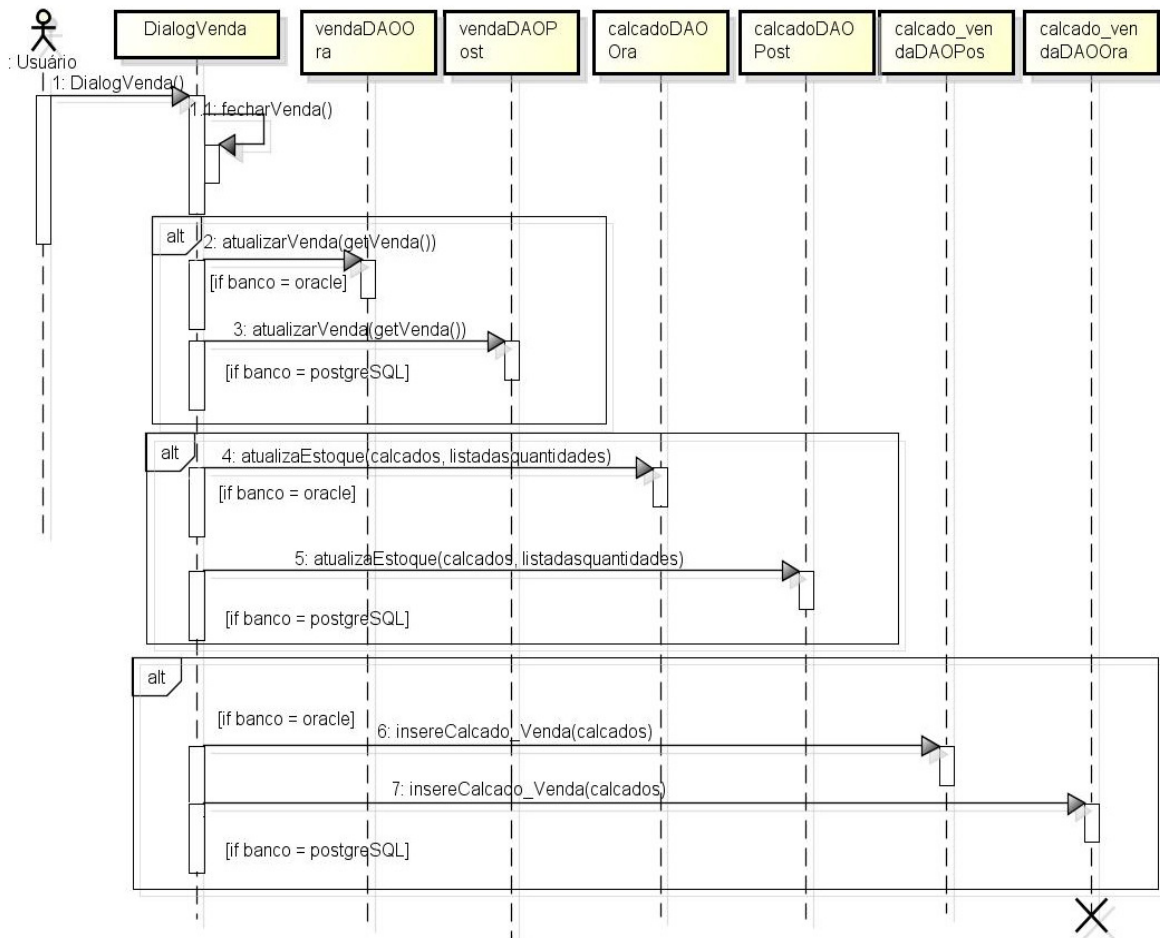


Figura 20 - Diagrama de Sequencia referente a realização de uma venda
Fonte: Autoria Própria

Na Figura 20 é mostrado através do diagrama de sequencia como é o processo de venda. É possível dividi-lo em três partes:

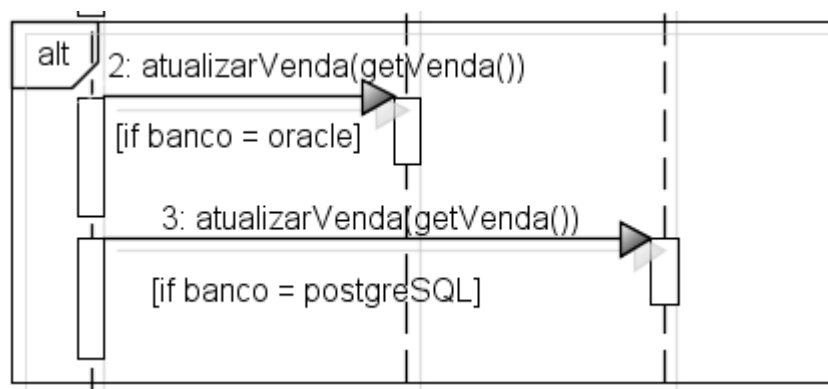


Figura 21 - Trecho onde é chamado o procedimento atualizarVenda()
Fonte: Autoria Própria

A Figura 21 mostra onde é chamado o procedimento `atualizarVenda()` na classe `VendaDAO` respectiva ao banco de dados escolhido. É passado como parâmetro os dados da venda, que incluem o id, o funcionário que a realizou, o id do cliente, o valor da comissão, a data e o valor total da venda. Neste procedimento é chamada a *procedure* `prc_atualizar_venda`.

A segunda etapa da venda se dá no momento de atualização de estoque.

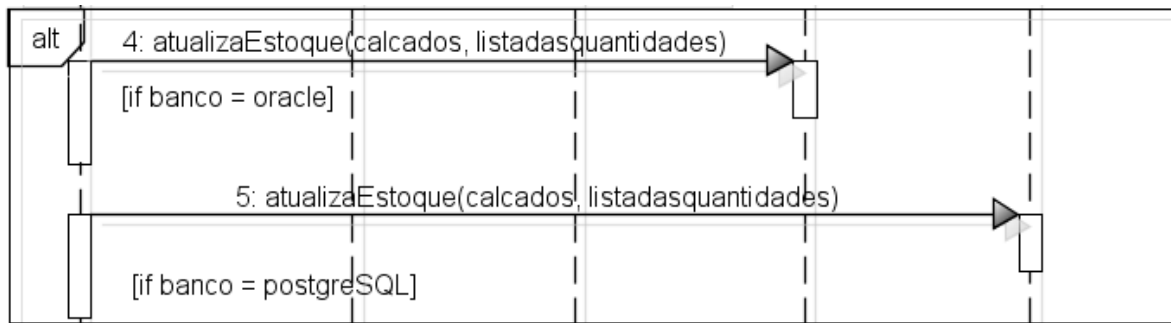


Figura 22 - Trecho do diagrama exibindo a execução do método atualizaEstoque()
 Fonte: Autoria Própria

Na Figura 22 é possível ver que o método executado é o atualizaEstoque(), ele está localizado na classe CalcadoDAO do seu respectivo banco de dados. Esse procedimento recebe como parâmetro a lista de calçados que o cliente comprou e a lista contendo a quantidade deles que foram compradas. Neste procedimento é chamada a *procedure* call prc_atualizar_calcado.

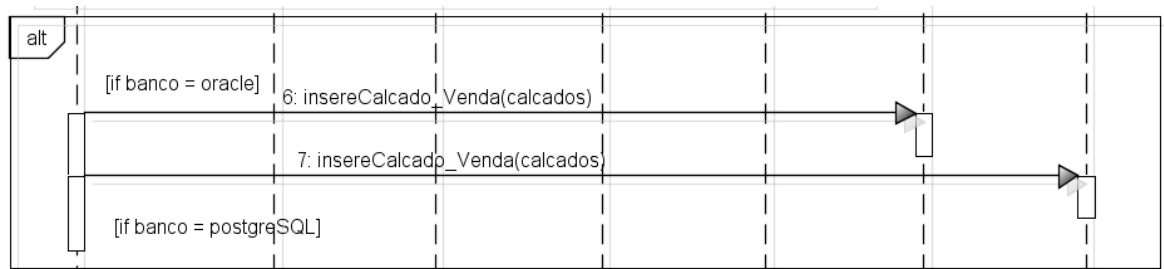
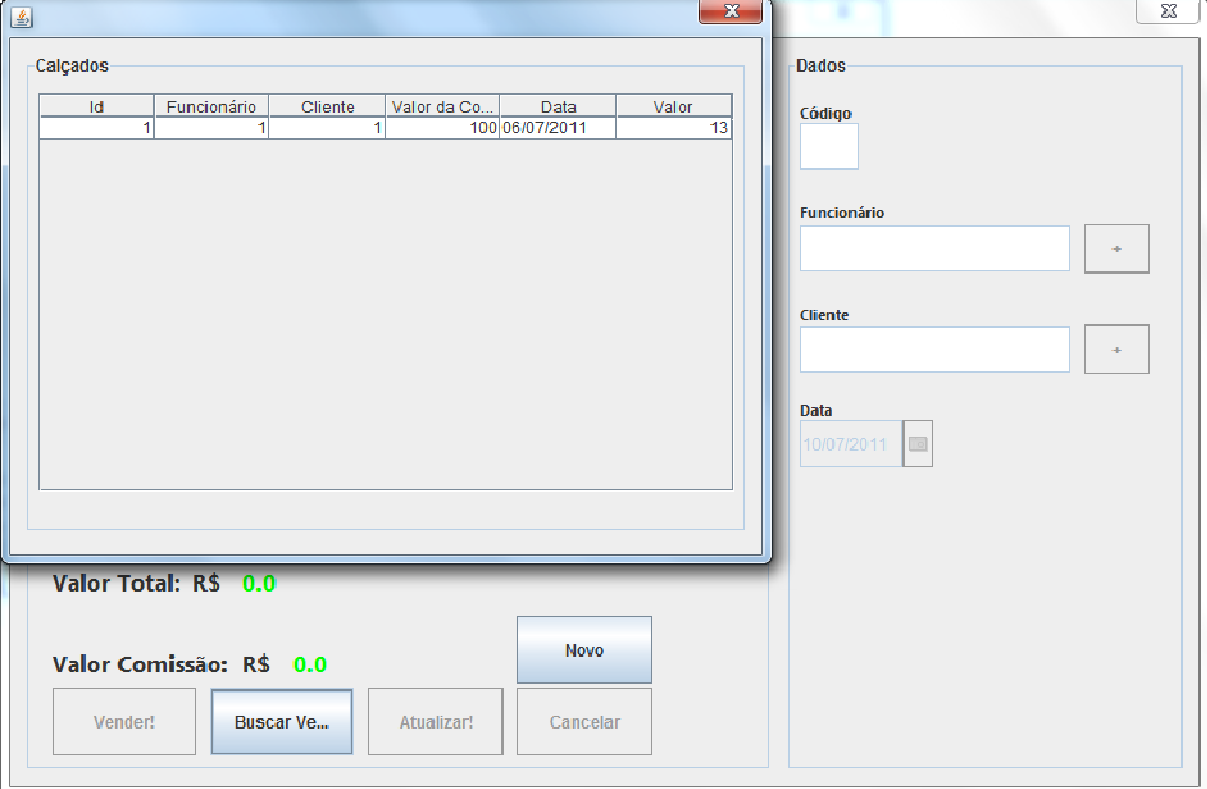


Figura 23 - Trecho do diagrama exibindo o chamamento do método insereCalcado_Venda()
 Fonte: Autoria Própria

A Figura 23 mostra a terceira e última parte do processo de venda, onde a é preenchida a tabela associativa responsável por relacionar os calçados com a venda. O método insereCalcado_Venda() chama a *procedure* prc_insere_calcado_venda na classe Calcado_VendaDAO respectivo ao banco de dados escolhido.

A Figura 24 exibe a tela com as todas as vendas cadastradas, essa tela é exibida quando se clica no botão “Buscar Venda...”. Esta tela é utilizada para que seja escolhida uma venda para alteração.



Id	Funcionário	Cliente	Valor da Co...	Data	Valor
1	1	1	100	06/07/2011	13

Valor Total: R\$ 0.0

Valor Comissão: R\$ 0.0

Buttons: Vender!, Buscar Ve..., Atualizar!, Cancelar, Novo

Dados:

Código:

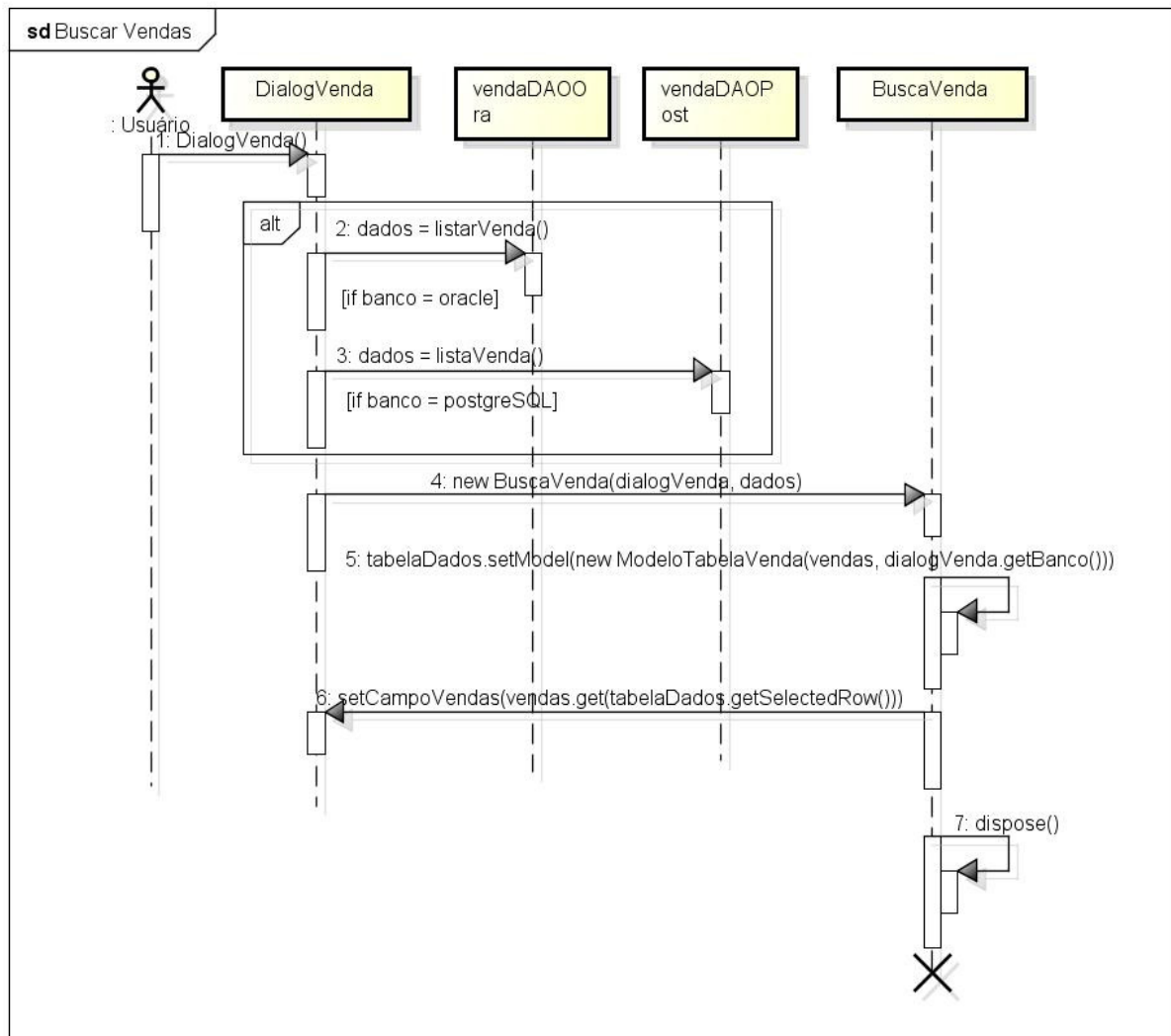
Funcionário: +

Cliente: +

Data: 10/07/2011

Figura 24 - Tela de venda exibindo todas as vendas registradas no banco
Fonte: Autoria Própria

Na Figura 25, o diagrama exibe os passos para a recuperação de uma venda.



powered by astah®

Figura 25 - Diagrama de sequencia relativo a venda

Fonte: Autoria Própria

Primeiro a aplicação acessa o banco de dados através do método listarVenda() na classe vendaDAO, esse método chama a função fnc_listar_venda que por sua vez retorna uma lista de vendas. Essa lista então é passada como parâmetro na criação da tela BuscaVenda(), que a utiliza para montar a tabela com os registros e exibi-los para o usuário. Após o usuário escolher qual venda deseja recuperar, a tela DialogVenda() preenche os campos dos dados da venda e monta a tabela com os calçados envolvidos na negociação.

Quando uma venda é recuperada, as opções do usuário são as mesmas de quando ele cadastrou uma venda nova. Pode-se alterar o cliente, o vendedor, alterar a data e adicionar ou remover calçados.

A Figura 26 exibe a tela DialogVenda() com a venda alterada. No caso foram adicionados dois calçados à venda.

Calçados da Venda

Id	Marca	Modelo	Valor	Tamanho	Quantidade
5	CONVERSE	ALL-STAR	100	40	1
4	KICHUTE	VAGAL	30	42	1
1	ADIDAS	SUPERSTAR	200	40	1

Dados

Código
1

Funcionário
A

Cliente
JOSÉ CLIENTE

Data
06/07/2011

Valor Total: R\$ 330.0

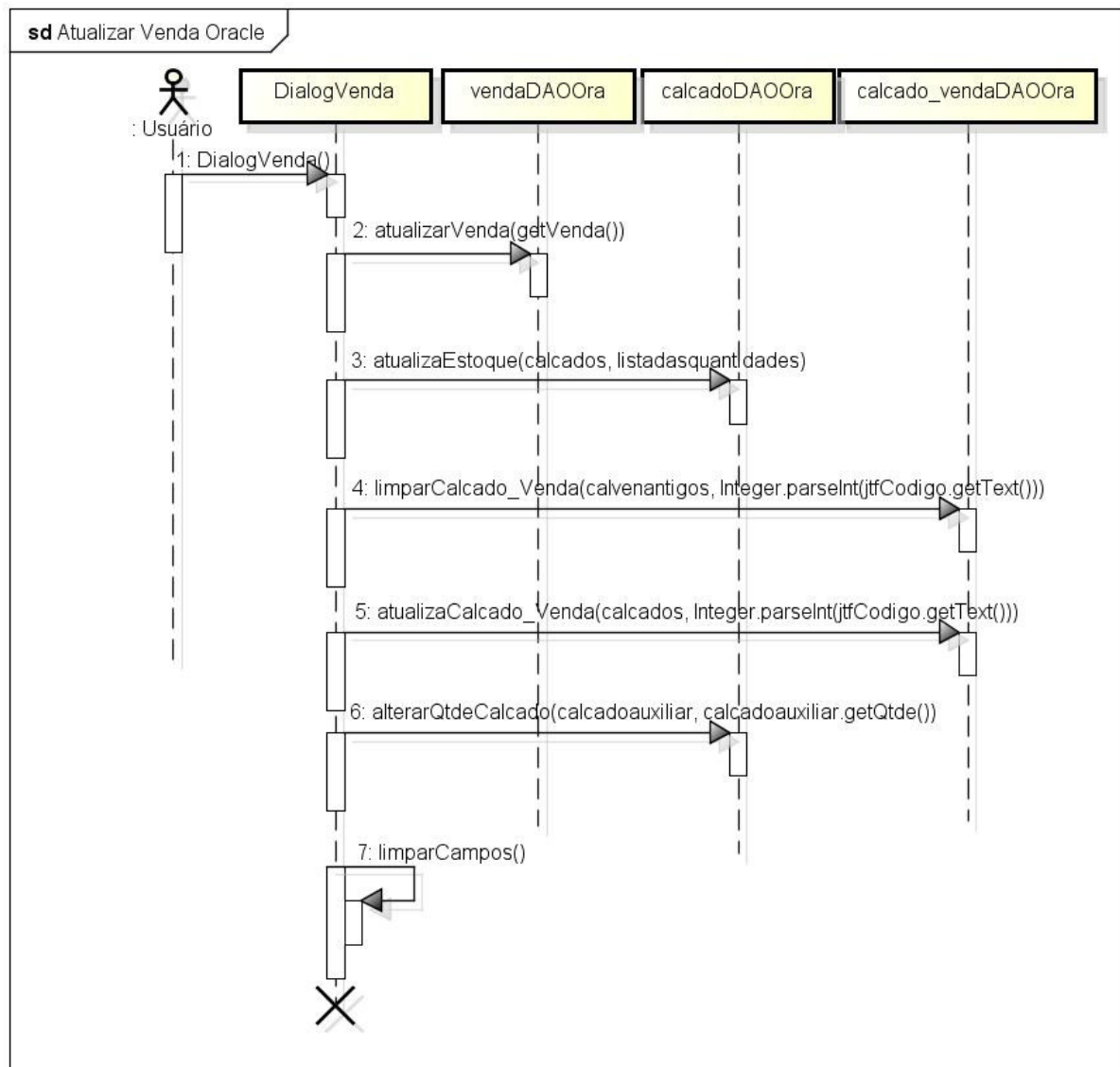
Valor Comissão: R\$ 42.9

Novo

Vender! Buscar Ve... Atualizar! Cancelar

Figura 26 - Tela de venda com uma venda prestes a ser atualizada
Fonte: Autoria Própria

No diagrama de sequencia na Figura 27 o processo de atualização de uma venda no banco Oracle é explicado.



powered by astah®

Figura 27 - Diagrama de sequencia descrevendo como é feita uma atualização de venda no Oracle

Fonte: Autoria Própria

Através do método `getVenda()` a tela `DialogVenda` recupera a venda que está na tela. Da mesma, o terceiro método `atualizaEstoque` recupera quais calçados estão nessa venda que foi atualizada e qual a quantidade deles. Os métodos seguintes, `limparCalçado_Venda` e `atualizaCalçado_Venda`, são utilizados para atualizar os dados da tabela associativa `tb_calçado_venda`. O último método, `alterarQtdeCalçado` faz o controle do estoque de calçados na tabela `tb_calçado`. O processo para banco PostgreSQL é o mesmo, trocando-se apenas as classes do Oracle para sua versão PostgreSQL.

5 TESTES DE DESEMPENHO

Para a realização dos testes de desempenho, foi utilizada a ferramenta P6Spy auxiliado pelo SQL Profiler v0.3.

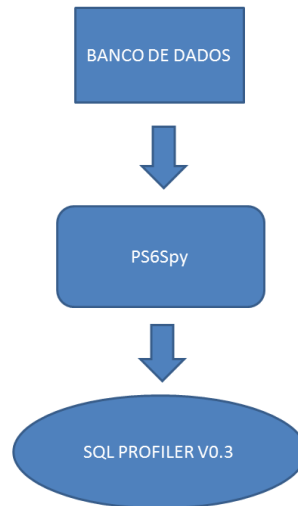


Figura 28 - Estrutura dos testes de desempenho
Fonte: Autoria Própria

A Figura 28 mostra como é esquema utilizado para os testes de desempenho. O SQL Profiler se conecta ao P6Spy, responsável monitorar as instruções executadas no banco de dados, e exibe os resultados em tempo real. Na Figura 29 é exibida a tela do SQL Profiler com uma requisição feita ao banco de dados capturada.

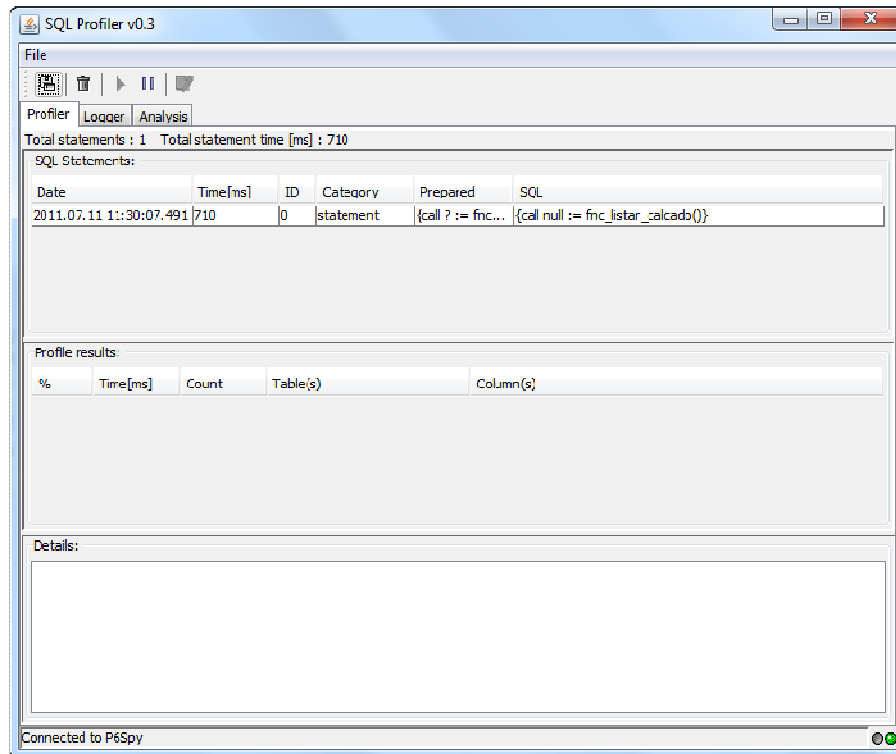


Figura 29 - Tela do SQL Profiler v0.3 com uma requisição capturada
Fonte: Autoria Própria

Na Figura 29 é possível ver informações referentes a requisição como por exemplo o tempo que levou e a hora que foi feita a requisição. Baseado nessa medida de tempo os gráficos referentes aos testes de desempenho foram criados.

Todas as funções e *procedures* testadas foram medidas nos dois bancos de dados, primeiro foram testadas com 10 repetições, depois com 30 repetições e por último com 60 repetições. O proposito disso foi a verificação de variações a medida que os teste eram repetidos.

A primeira bateria de testes foi referente a consultas, mais especificamente a função `fnc_listar_calçado()`.

5.1 TESTES DA FUNÇÃO FNC_LISTAR_CALCADO()

5.1.1 Testes da função FNC_LISTAR_CALCADO – Oracle

Inicialmente foram realizados os testes sobre a função `FNC_LISTAR_CALCADO`, versão criada no banco de dados Oracle.

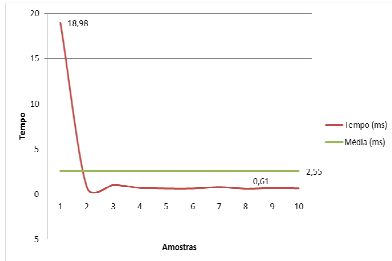


Gráfico 1 - fnc_listar_calçado - 10 repetições – Oracle
Fonte: Autoria Própria.

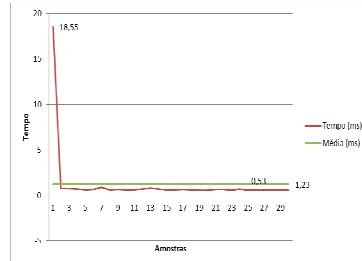


Gráfico 2 - fnc_listar_calçado - 30 repetições – Oracle
Fonte: Autoria Própria.

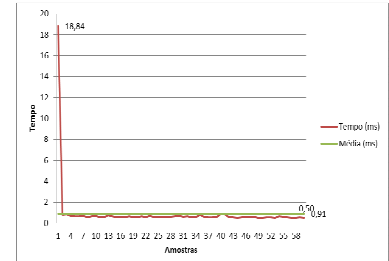


Gráfico 3 - fnc_listar_calçado - 60 repetições – Oracle
Fonte: Autoria Própria.

Os testes mostraram que sempre o primeiro resultado era o que levava mais tempo para ser processado e por saírem da média geral influenciavam um pouco no resultado dela.

5.1.2 Testes da função FNC_LISTAR_CALCADO – PostgreSQL

A segunda parte dos testes foram realizados sobre a função FNC_LISTAR_CALCADO, versão criada no banco PostgreSQL.

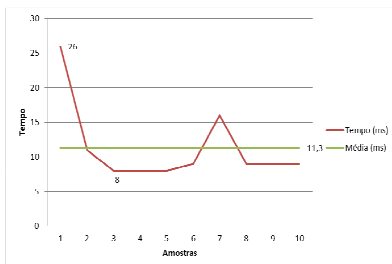


Gráfico 4 - fnc_listar_calçado - 10 repetições – PostgreSQL
Fonte: Autoria Própria.

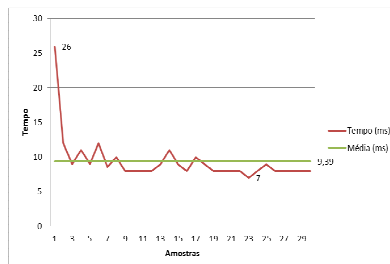


Gráfico 5 - fnc_listar_calçado - 30 repetições – PostgreSQL
Fonte: Autoria Própria.

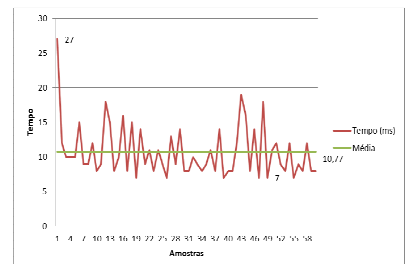


Gráfico 6 - fnc_listar_calçado - 60 repetições – PostgreSQL
Fonte: Autoria Própria.

Os testes no PostgreSQL demonstraram que houveram grandes variações entre uma repetição e outra. No teste com 60 repetições, em nenhum momento o tempo de resposta tendeu-se a estabilizar.

Assim como no Oracle, a primeira repetição sempre foi a que levou mais tempo, porém a diferença entre o resultado com tempo mais alto para os demais não foi tão grande.

5.1.3 Resultados dos testes na função fnc_listar_calçado

Os resultados mostraram que o tempo médio de resposta para o banco Oracle foi sempre inferior ao PostgreSQL, assim como os valores máximos e mínimos também foram inferiores. Os testes mostraram ainda que com o aumento

do número de repetições, a média de tempo de resposta do Oracle tendia a diminuir enquanto no PostgreSQL se manteve estável.

		10 amostras	30 amostras	60 amostras
Oracle	Média (ms)	2,55	1,23	2,36
	Coeficiente de Variação (%)	2,26	2,66	2,59
	Máximo (ms)	20	21	19
	Mínimo (ms)	1	0	0
PostgreSQL	Média (ms)	11,3	9,39	10,77
	Coeficiente de Variação (%)	0,50	0,35	0,34
	Máximo (ms)	26	36	27
	Mínimo (ms)	28	7	7

Tabela 1 - Resultados dos testes realizados na função fnc_listar_calçado

Fonte: Autoria Própria. Isso é uma tabela

5.2 TESTES DA PROCEDURE PRC_ATUALIZAR_FUNCIONARIO

Foram realizados sobre da *procedure* prc_atualizar_funcionario, com o objetivo de inserir novos registros.

5.2.1 Testes da *procedure* PRC_ATUALIZAR_FUNCIONARIO – Oracle

Primeiro foi testada a *procedure* PRC_ATUALIZAR_FUNCIONARIO em sua versão Oracle.

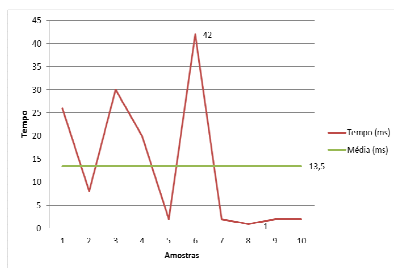


Gráfico 7- Testes prc_atualizar_funcionario - 10 repetições – Oracle
Fonte: Autoria Própria.

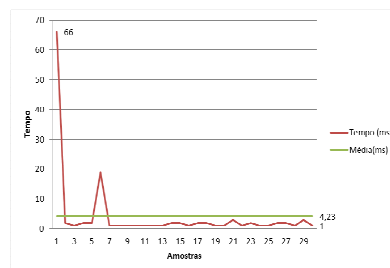


Gráfico 8 - Testes prc_atualizar_funcionario - 30 repetições – Oracle
Fonte: Autoria Própria.

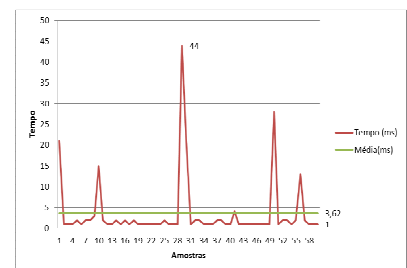


Gráfico 9 - Testes prc_atualizar_funcionario - 60 repetições – Oracle
Fonte: Autoria Própria.

O teste com 10 repetições mostrou uma grande variação entre uma repetição e outra, sendo que a repetição mais demorada foi a de número 6. No teste de 30 repetições o tempo tendeu a se estabilizar conforme as repetições iam acontecendo, demonstrando pouca variação. Já no teste de 60 repetições a variação foi maior sendo que o maior tempo foi alcançado na metade dos testes.

5.2.2 Testes da *procedure* PRC_ATUALIZAR_FUNCIONARIO – PostgreSQL

Testes sobre a função PRC_ATUALIZAR_FUNCIONARIO em sua versão do banco PostgreSQL

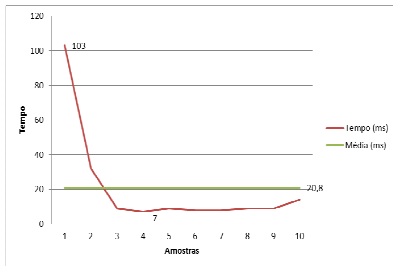


Gráfico 10 - Testes prc_atualizar_funcionario- 10 repetições – PostgreSQL
Fonte: Autoria Própria.

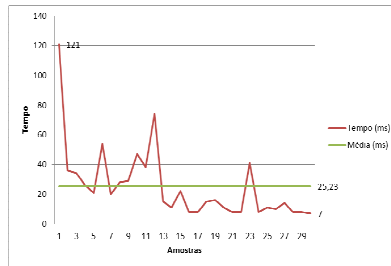


Gráfico 11 - Testes prc_atualizar_funcionario - 30 repetições – PostgreSQL
Fonte: Autoria Própria.

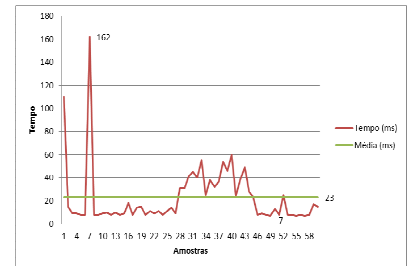


Gráfico 12 - Testes prc_atualizar_funcionario- 60 repetições - PostgreSQL
Fonte: Autoria Própria.

Enquanto que o teste com 10 repetições mostrou uma estabilidade no tempo de resposta, conforme o número de repetições aumentava o tempo passou a variar entre uma repetição e outra. É possível notar no Gráfico 12 que diferente dos dois primeiros testes, o tempo mais alto não foi na primeira repetição.

5.2.3 Resultados dos testes na *procedure* prc_atualizar_funcionario.

Os testes para a *procedure* prc_atualizar_funcionario apresentaram como resultado um quadro semelhante ao do teste anterior. Novamente o tempo de resposta do Oracle foi menor e sua média tendeu a baixar a medida que eram realizados os testes com mais repetições, enquanto no PostgreSQL a média se manteve parecida.

O tempo médio para a execução de uma *procedure* inserindo um registro novo foi maior que o tempo médio de uma listagem.

		10 amostras	30 amostras	60 amostras
Oracle	Média (ms)	13,4	4,23	3,62
	Coefficiente de Variação (%)	1,10	2,86	2,07
	Máximo (ms)	42	66	44
	Mínimo (ms)	1	1	1
PostgreSQL	Média (ms)	20,8	25,23	23
	Coefficiente de Variação (%)	1,43	0,96	1,13
	Máximo (ms)	103	121	162
	Mínimo (ms)	7	7	23

Tabela 2 - Resultados dos testes realizados na *procedure* prc_atualizar_funcionario

Fonte: Autoria Própria.

5.3 TESTES DA PROCEDURE PRC_ATUALIZAR_CLIENTE()

O terceiro teste envolveu a *procedure* PRC_ATUALIZAR_CLIENTE(), atualizando um registro existente.

5.3.1 Teste PRC_ATUALIZAR_CLIENTE - ORACLE

Primeiro foi testada a *procedure* PRC_ATUALIZAR_CLIENTE, na versão do banco Oracle.

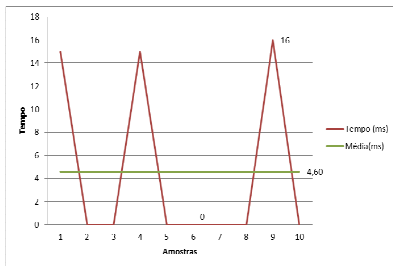


Gráfico 13 - Testes prc_atualizar_cliente – 10 repetições – ORACLE
Fonte: Autoria Própria.

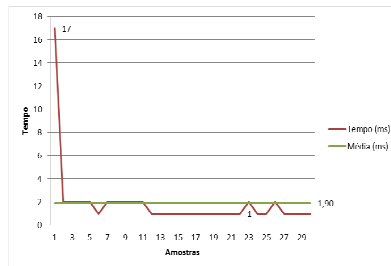


Gráfico 14 - Testes prc_atualizar_cliente – 30 repetições – ORACLE
Fonte: Autoria Própria.

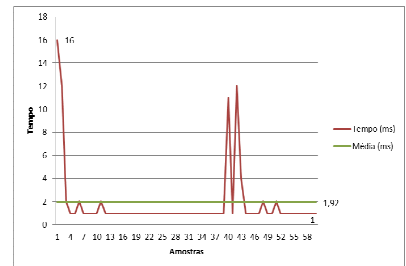


Gráfico 15 - Testes prc_atualizar_cliente – 60 repetições – ORACLE
Fonte: Autoria Própria.

No Gráfico 13 é mostrado que no teste com 10 repetições houve bastante alteração na variação do tempo, chegando próximo ao valor máximo duas vezes.

Com 30 e 60 repetições a variação foi menor, porém no Gráfico 15 é possível notar que após um pouco mais da metade do teste o tempo aumentou consideravelmente duas vezes antes de voltar ao valor próximo da média. O valor máximo foi atingindo logo na primeira repetição em ambos os testes.

5.3.2 Teste PRC_ATUALIZAR_CLIENTE – PostgreSQL

Os testes seguintes foram relativos a *procedure* PRC_ATUALIZAR_CLIENTE, na versão do banco de dados PostgreSQL.

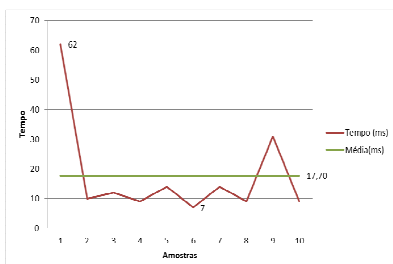


Gráfico 16 - Testes prc_atualizar_cliente – 10 repetições – PostgreSQL
Fonte: Autoria Própria.

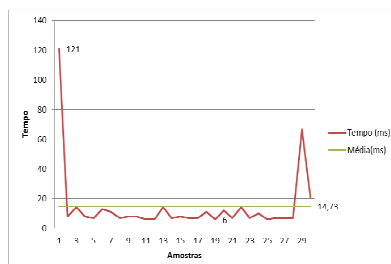


Gráfico 17 - Testes prc_atualizar_cliente – 30 repetições – PostgreSQL
Fonte: Autoria Própria.

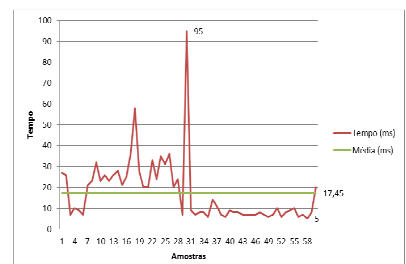


Gráfico 18 - Testes prc_atualizar_cliente – 60 repetições – PostgreSQL
Fonte: Autoria Própria.

No PostgreSQL o teste com 10 repetições apresentou uma variação menor em relação ao mesmo teste no Oracle.

No Gráfico 18 é possível notar uma grande variação no tempo entre o começo e o fim do teste, culminando com o resultado mais demorado perto da metade das repetições. É possível perceber ainda nesse teste como o tempo passou a trabalhar abaixo da média após o valor mais alto.

5.3.3 Resultados dos testes na *procedure* *prc_atualizar_cliente*

Os tempos de resposta relativos ao Oracle novamente foram menores que os do banco PostgreSQL. Enquanto no primeiro banco a média abaixou em relação ao teste de 10 para 30 amostras e teve um leve aumento quando comparadas as médias dos testes de 30 e 60 amostras, o segundo banco de dados já apresentou uma mudança maior onde média relativa ao teste de 60 amostras foi parecida com a do primeiro teste, com 10 amostras.

O coeficiente de variação do banco Oracle teve pouca mudança enquanto no PostgreSQL os valores variaram mais. Outro ponto a ser notado foi a grande variação no valor máximo dos testes do banco PostgreSQL.

		10 amostras	30 amostras	60 amostras
Oracle	Média (ms)	4,60	1,90	1,92
	Coeficiente de Variação (%)	1,61	1,52	1,56
	Máximo (ms)	16	17	16
	Mínimo (ms)	1	1	1
PostgreSQL	Média (ms)	17,70	14,73	17,45
	Coeficiente de Variação (%)	0,96	1,55	0,86
	Máximo (ms)	62	121	95
	Mínimo (ms)	7	6	5

Tabela 3 - Resultados dos testes realizados na *procedure* *prc_atualizar_cliente*

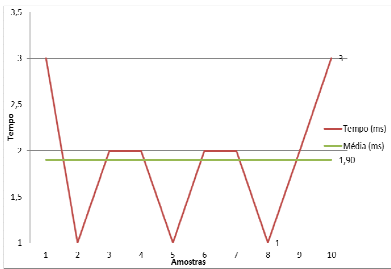
Fonte: Autoria Própria.

5.4 TESTES DA *PROCEDURE* *PRC_APAGA_FUNCIONARIO*

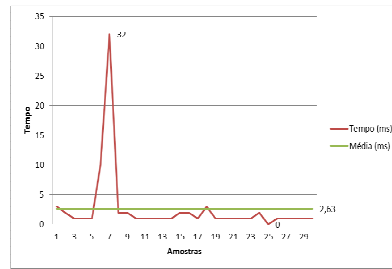
Os últimos testes envolveram a *procedure* *PRC_APAGA_FUNCIONARIO*.

5.4.1 Teste *PRC_APAGA_FUNCIONARIO* – Oracle

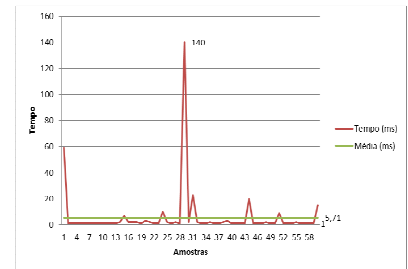
Primeiro foi testada a *procedure* *PRC_APAGA_FUNCIONARIO*, na versão do banco de dados Oracle.



**Gráfico 19 - Testes
prc_apagar_funcionario – 10
repetições – Oracle
Fonte: Autoria Própria.**



**Gráfico 20 - Testes
prc_apagar_funcionario – 30
repetições – Oracle
Fonte: Autoria Própria.**



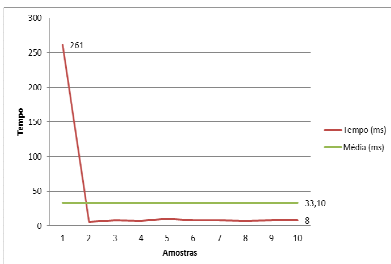
**Gráfico 21 - Testes
prc_apagar_funcionario – 60
repetições – Oracle
Fonte: Autoria Própria.**

O Gráfico 19 mostra que houve uma grande variação do tempo de resposta durante o teste de 10 repetições, sendo o que o tempo máximo foi atingindo duas vezes ao longo das repetições.

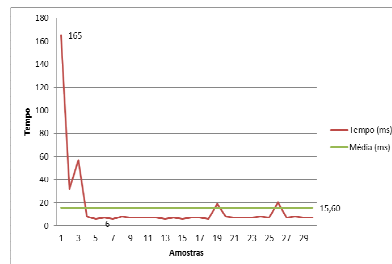
Os testes com 30 e 60 repetições demonstraram-se mais estáveis com exceção do valor máximo que foi bem mais alto do que a média.

5.4.2 Teste PRC_APAGAR_FUNCIONARIO – POSTGRESQL

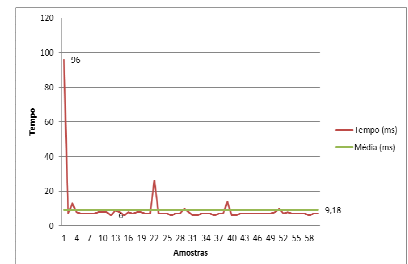
Os testes realizados sobre a *procedure* PRC_APAGAR_FUNCIONARIO, na versão do banco PostgreSQL.



**Gráfico 22 - Testes
prc_apagar_funcionario – 10
repetições – PostgreSQL
Fonte: Autoria Própria.**



**Gráfico 23 - Testes
prc_apagar_funcionario – 30
repetições – PostgreSQL
Fonte: Autoria Própria.**



**Gráfico 24 - Testes
prc_apagar_funcionario – 60
repetições – PostgreSQL
Fonte: Autoria Própria.**

Os testes na *procedure* PRC_APAGAR_FUNCIONARIO apresentaram resultados parecidos no sentido de variação do tempo conforme pode ser visto nos Gráficos 22, 23 e 24. Nos três testes o tempo de resposta mais alto foi logo na primeira repetição e conforme as repetições seguintes eram executadas, os tempos passaram a diminuir com alguma variação ocasional.

5.4.3 Resultados dos testes na *procedure* `prc_apagar_funcionario`

Nos testes relativos a *procedures* de exclusão de registros, o tempo de resposta do Oracle também foi inferior ao do PostgreSQL. Nestas *procedures* em particular foi onde houveram as maiores diferenças entre o tempo médio de resposta entre um banco e outro.

Estes testes apresentaram um quadro particular de resposta. Enquanto nos testes das outras duas *procedures* e função a média do Oracle tendeu a diminuir, nestes testes a média tendeu a aumentar, assim como seus valores máximos e seus coeficientes de variação. No PostgreSQL o resultado foi atípico também, onde a média tendeu-se a diminuir, assim como seus valores máximos, mínimos e os coeficientes de variação.

		10 amostras	30 amostras	60 amostras
Oracle	Média (ms)	1,90	2,63	5,71
	Coeficiente de Variação (%)	0,38	2,20	3,32
	Máximo (ms)	3	32	140
	Mínimo (ms)	1	0	1
PostgreSQL	Média (ms)	33,10	14,73	9,18
	Coeficiente de Variação (%)	2,41	1,92	1,27
	Máximo (ms)	261	121	96
	Mínimo (ms)	8	6	6

Tabela 4 - Resultados dos testes realizados na *procedure* `prc_apagar_funcionario`

Fonte: Autoria Própria.

6 CONSIDERAÇÕES FINAIS

6.1 CONCLUSÃO

Uma das conclusões que se pode tirar desse trabalho é a capacidade das linguagens procedurais, que adicionam inúmeros recursos para se trabalhar com uma base dados, sendo indispensáveis para qualquer DBA. Mesmo o Oracle apresentando mais recursos na sua linguagem PL/SQL é possível através de adaptações trazer os mesmos recursos para o PL/pgSQL.

A versão reduzida do Oracle, o 10g XE é uma ótima ferramenta de estudo e uma boa solução para quem deseja trabalhar com o Oracle sem precisar utilizar sua ferramenta completa que pode se tornar pesada e inviável dependendo do objetivo do projeto.

A Tabela 5 traz alguns dados relativos aos testes de desempenho que merecem observações.

		Resultados
Oracle	Média Geral (ms)	3,38
	Maior Máximo (ms)	140
	Menor Mínimo (ms)	0
	Maior Coeficiente de Variação (%)	3,32
	Menor Coeficiente de Variação (%)	0,38
PostgreSQL	Média Geral (ms)	16,25
	Maior Máximo (ms)	261
	Menor Mínimo (ms)	5
	Maior Coeficiente de Variação (%)	2,41
	Menor Coeficiente de Variação (%)	0,34

Tabela 5 - Tabela de resultados específicos

Fonte: Autoria Própria

A média geral do Oracle foi bem inferior à média do PostgreSQL, seu maior valor máximo e seu menor valor mínimo também foram menores. A variação do Oracle foi maior já que tanto seu maior coeficiente de variação como o seu menor foram maiores que os do banco PostgreSQL demonstrando que o último tende a se manter estável conforme é executado não alterando sua velocidade tanto para mais quanto para menos.

Os testes demonstraram que no caso estudado o desempenho do Oracle 10g XE e sua linguagem PL/SQL foi sempre superior ao PostgreSQL 8.4 e a sua linguagem PL/pgSQL mas é válido lembrar que este é um banco de dados livre de código aberto mantido por uma comunidade e mesmo assim é um banco bastante

robusto, com muitos recursos e em constante evolução, sendo uma ótima alternativa para quem busca um banco *open-source* para trabalhar com seus dados.

Ambos os bancos de dados são de fácil instalação, a integração deles com o Java é bastante simples devido as bibliotecas fornecidas pelos criadores. Eles possuem bastante documentação disponível e uma comunidade madura e ativa. Livros, dicas e informações estão disponíveis em grande quantidade pela Internet facilitando muito o trabalho de quem deseja se aprofundar na área de banco de dados.

6.2 TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO

Os sistemas gerenciadores de bancos de dados abordados neste trabalho estão em constante atualização. O PostgreSQL nas suas ultimas versões recebeu mais recursos que antes só haviam no Oracle, diminuindo assim um pouco da necessidade de fazer grandes adaptações no código quando se queria passar de um banco para o outro. A tendência é que o PostgreSQL contenha mais recursos do Oracle a medida que for atualizado, melhorando assim o seu desempenho e capacitando ainda mais o seu banco de dados.

Os testes de desempenho feitos no trabalho não são verdades absolutas, porém já permitem traçar um parâmetro comparativo básico de desempenho.

A área de banco de dados é uma das mais importantes na área de tecnologia da informação. A todo momento são trazidas e inventadas novas tecnologias e estudos como o deste trabalho são importantes por trazer ideias e conceitos para essa área interessando a todos

REFERÊNCIAS BIBLIOGRÁFICAS

1KEYDATA. **SQL Constraint**. 2011. Disponível em <<http://www.1keydata.com/sql/sql-constraint.html>>. Acessado em 02 de Agosto de 2011.

CHAMBERLIN, Don. **Encyclopedia of Database Systems Volume I**. Nova York, Springer Science+Businnes Media, 2009.

DEMOISELE. **Documentação Persistence**. 2010. Disponível em: <http://www.frameworkdemoiselle.gov.br/modulo/framework-persistence/persistence/docs#Criação_de_Classe_POJO>. Acessado em 4 de Julho de 2011.

DOEDERLEIN, Osvaldo Pinali. **Oracle para desenvolvedores Java**. 2009. Disponível em <<http://www.devmedia.com.br/post-8620-Artigo-Java-Magazine-43-Oracle-para-Desenvolvedores-Java.html>>. Acessado em 3 de Agosto de 2011.

EEMBLEY, W, David. **Encyclopedia of Database Systems Volume I**. Nova York, Springer Science+Businnes Media, 2009.

ESPAÇOINFO. **SGBD: O que é?**. 2011. Disponível em <<http://espacoinfo.net/o-que-e-sgbd-bd-ii/>> Acessado em 02 de Março de 2011.

FEUERSTEIN, Steven; PRIBYL, Bill; **Oracle PL/SQL Programming 5th Edition**. 2009. O'Reilly Media Inc, Sebastopol.

FERRARI, Fabrício. **Crie Banco de Dados em MySQL**. 2007. São Paulo. Digerati Books.

FOSTER, Ian. **O que é a Computação em Grelha (Grid Computing)?**. 2011. Disponível em <<http://visibilidade.net/tutorial/computacao-Grid.html>> . Acessado em 06 de Maio de 2011.

GOYA, Milton; **PL/SQL - Procedures e Funções**. 2004. Disponível em: <<http://www.linhadecodigo.com.br/ArtigoImpressao.aspx?id=335>> Acessado em 16 de Abril de 2011.

HELLAND, Pat. **Encyclopedia of Database Systems Volume I**. Nova York, Springer Science+Businnes Media, 2009.

JAVA WIKI. **Java**. 2011. Disponível em: <<http://java.wikia.com/wiki/Java>> Acessado em 29 de Julho de 2011.

JAVAFREE. **Tutorial Java: O que é Java?**. 2009. Disponível em: <<http://javafree.uol.com.br/artigo/871498/>>. Acessado em 23 de Junho de 2011.

KALLAS, César. PUC – Campinas Apostila de PLSQL. 2008. Disponível em: <www.cesarkallas.net/arquivos/faculdade/banco_dados2/PLSQL/Apostila_PLSQL.doc>. Acessado em 20 de Abril de 2011.

MENEZES, ANGÉLO; BARROSO, LISBÂNIO; TAXIMAN, SERIMAMIS; ALVES, VAGNER; CARODOSO, ANTÔNIO. **Banco de Dados – Uma visão geral**. 2008. Disponível em < <http://pt.scribd.com/doc/15262732/Banco-de-Dados-Uma-Visao-Geral>> Acessado em 02 de Março de 2011.

MOJIAN, Bruce. **PostgreSQL: Introduction and Concepts**. 2001. Pearson Education, Upper Saddle River.

NOVELLI, Márcio. **SGBD - Sistema Gerenciador de Banco de Dados**. 2006. Disponível em < <http://www.plugmasters.com.br/sys/materias/108/1/SGBD---Sistema-Gerenciador-de-Banco-de-Dados>> Acessado em 02 de Março de 2011.

ORACLE. **Oracle® Database PL/SQL User's Guide and Reference**. 2005. Disponível em <http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14261/overview.htm#LNPLS001> . Acessado em 04 de Julho de 2011.

ORACLE. **Introdução ao Oracle: SQL e PL/SQL**. 2000. Oracle do Brasil Sistemas Ltda. São Paulo.

ORACLE. **Oracle8i Concepts**. 1999. Disponível em < <http://www.cs.umbc.edu/portal/help/oracle8/server.815/a67781/c08schem.htm>> . Acessado em 05 de Julho de 2011.

ORACLE. **Oracle Database 10g Release 2 JDBC Drivers**. 2011. Disponível em <<http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-10201-088211.html>>. Acessado em 26 de Junho de 2011.

ORACLE SUN DEVELOPER NETWORK. **The Java Language Environment**. 2010. Disponível em: <http://java.sun.com/docs/white/langenv/Intro.doc2.html>. Acesso em: 15 Junho de 2011.

PEREZ, F. C. LUÍS. **Avaliação de produtividade PHP X Java no desenvolvimento de sistemas de software**. 2010. Disponível em < http://www.ricardoterra.com.br/publications/students/2010_perez.pdf > Acessado em 03 de Março de 2011.

POSTGRESQL GLOBAL DEVELOPMENT GROUP. **Documentação do PostgreSQL 8.0.0**. 2005. Disponível em < <http://pgdocptbr.sourceforge.net/pg80/plpgsql.html> >. Acessado em 02 de Março de 2011

POSTGRESQL GLOBAL DEVELOPMENT GROUP. **PostgreSQL 8.4.8 Documentation.** 2009. Disponível em < <http://www.postgresql.org/docs/8.4/static/index.html>>. Acessado em 02 de Abril de 2011

POSTGRESQL GLOBAL DEVELOPMENT GROUP. **JDBC Driver.** 2009. Disponível em < <http://jdbc.postgresql.org/download.html>>. Acessado em 27 de Junho de 2011

RAMKRISHNAN, Raghu. **Database Management Systems Third Edition.** Cingapura, McGraw-Hill Higher Education, 2003.

RIBEIRO, Luis. **Principais Bancos de Dados Relacionais.** 2010. Disponível em < <http://www.artigonal.com/tecnologias-artigos/principais-bancos-de-dados-relacionais-2146567.html>> Acessado em 03 de Março de 2011.

SOUZACE. **Backup do Banco de Dados Oracle.** 2011. Disponível em < <http://pt.scribd.com/doc/50708081/Backup-do-Banco-de-Dados-Oracle>>. Acessado em 25 de Junho de 2011.

SINTECTUS. **Lição 1 - Introdução ao JDBC.** 2011. Disponível em: < <http://wiki.sintectus.com/bin/view/GrupoJava/LicaoIntroducaoAoJDBC>>. Acessado em 25 de Junho de 2011.

SS64. **Oracle Datatypes.** 2011. Disponível em: < <http://ss64.com/ora/syntax-datatypes.html>>. Acessado em 28 de Junho de 2011.

VEIRA, Sidney. **Tecnologia em Banco de Dados.** 2011. Disponível em: < http://sidneyviera.kinghost.net/abas/disciplinas/download/ES_Tecnologia_BD_Cursos.pdf>. Acessado em 1º Julho de 2011.