UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

FERRAMENTA DE APOIO A REESTRUTURAÇÃO DE CÓDIGO FONTE EM LINGUAGEM PL/SQL BASEADO EM PADRÕES DE LEGIBILIDADE

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

DYCKSON DYORGIO DOLLA

BLUMENAU, JUNHO/2001

2001/1-24

FERRAMENTA DE APOIO A REESTRUTURAÇÃO DE CÓDIGO FONTE EM LINGUAGEM PL/SQL BASEADO EM PADRÕES DE LEGIBILIDADE

DYCKSON DYORGIO DOLLA

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Everaldo Artur Grahl — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Everaldo Artur Grahl

_____

Prof. Oscar Dalfovo

_____

Prof. Wilson Pedro Carli

SUMÁRIO

RESUMO

Este trabalho consiste no desenvolvimento de uma ferramenta que analisa e faz a reestruturação e documentação interna de código fonte em PL/SQL, utilizando padrões de legibilidade obtidos em pesquisa bibliográfica. O protótipo especificado e desenvolvido neste trabalho efetua documentação dos programas e subrotinas, gera avisos de determinadas construções de programação e efetua a formatação de atributos do código-fonte como por exemplo a indentação, palavras-chave, colunas de tabelas e variáveis internas.

ABSTRACT

This study consists in a development tool that analyzes and restructures PL/SQL source codes, using legibility patterns, obtained in library research. The prototype specified and developed in this work performs documentation of the programs and subroutines, shows warnings about some programming constructions and format the attributes of the source code like indentation, keywords, table columns and internal variables.

LISTA DE FIGURAS

LISTA DE QUADROS

GLOSSÁRIO

ANSI	American National Standards
BFILE	Binary File
BLOB	Binary Large Object
CASE	Computer Aided Software Design
CLOB	Character Large Object
ISO	International Standards Organization
LOB	Large Object
NASA	National Aeronautics & Space Administration
NLOB	National Character Large Object
PL/SQL	Procedural Language / Structured Query Language
SGBD	Sistema Gerenciador de Banco de Dados
SQL	Structured Query Language
UML	Unified Modelling Language

1 INTRODUÇÃO

No processo de construção e manutenção de software os principais aspectos que devem ser observados são a legibilidade, documentação interna no código fonte e a maneira como os aplicativos são construídos em relação à lógica. Segundo Arthur (1985), a manutenção consome recursos vitais para o desenvolvimento de novos projetos. Sendo assim, o ideal é que este processo de manutenção seja o mais rápido e eficiente possível.

O desenvolvedor, ao seguir um padrão para nomenclatura e utilização das estruturas de dados, indentação e documentação de procedimentos internos, alcança um nível muito maior de confiabilidade e segurança nas manutenções realizadas, assim como a rapidez com que as mesmas poderão são realizadas.

A linguagem de consulta estruturada Structured Query Language (SQL) foi disponibilizada comercialmente pela primeira vez em 1979, pela empresa Relational Software Inc., que hoje é conhecida como ORACLE. Hoje em dia, segundo Oracle (1997a), a linguagem SQL é reconhecida e aceita como o padrão de linguagem para bancos de dados relacionais.

Uma das maiores qualidades do SQL é a de beneficiar um grande número de usuários, deste programadores de aplicação até os usuários finais. O principal objetivo do SQL é o de apresentar uma interface para acessar bancos de dados relacionais como Oracle, e todos os comandos SQL são na verdade instruções para o banco de dados.

PL/SQL é a extensão procedural do SQL, criado pela ORACLE. A linguagem de programação bloco estruturada PL/SQL, segundo Oracle (1997b), oferece características modernas de engenharia, como tratamento de exceções e encapsulamento de dados. A linguagem de programação PL/SQL também oferece uma integração completa ao servidor Oracle e suas ferramentas, assim como portabilidade e segurança, combinando assim a facilidade de manipulação de dados que a linguagem SQL oferece, com processamento de dados das linguagens procedurais.

Para implementar um padrão em aplicativos já construídos, é necessário submetê-los a um processo de reestruturação de suas estruturas internas, a fim de atender os padrões e

normas desejados. O processo de reestruturação, segundo Furlan (1994), pode ser definido como um processo de análise de fluxos de controle e lógica de programação com a geração de uma versão estruturada do código fonte original sem alteração de sua funcionalidade.

Com a construção de uma ferramenta que utilize algum tipo de padronização ou metodologia de qualidade para formatar código fonte, pode-se alcançar melhores resultados no processo de manutenção e correção de software.

Foi redigido por Dalmolin (2000), a proposta para uma ferramenta de reestruturação de código fonte da linguagem de programação C/C++, onde o conceito do trabalho é basicamente o mesmo apresentando aqui, ou seja, baseando-se em um conjunto de padrões e normas de legibilidade, é construída uma ferramenta que realiza este processo automaticamente.

A escolha do tema deste trabalho, uma ferramenta para reestruturar código fonte PL/SQL, veio justamente da sugestão apresentada no trabalho de Dalmolin (2000), de realizar o processo para outras linguagens de programação disponíveis no mercado.

1.1 OBJETIVO

Este trabalho tem como objetivo principal especificar e implementar uma ferramenta que apóie a reestruturação de código fonte escrito em linguagem PL/SQL utilizando padrões de legibilidade. Estes padrões foram obtidos a partir de pesquisa bibliográfica.

Os objetivos específicos a serem alcançados na execução deste protótipo são os seguintes:

a) Efetuar a indentação do código fonte;

b) Incluir documentação em pontos-chave do código fonte;

c) Demonstrar avisos sobre determinadas construções não-recomendadas;

d) Formatar ou modificar estruturas para atender os requisitos de legibilidade estudados.

1.2 ORGANIZAÇÃO

O trabalho é composto por quatro capítulos. Neste primeiro capítulo foram apresentados a origem do trabalho, objetivo e organização.

No segundo capítulo são apresentadas as principais características da linguagem estruturada de consulta SQL e a linguagem de programação PL/SQL.

No terceiro capítulo são apresentados também conceitos e características da qualidade de software e os padrões utilizados para a elaboração do software.

No quarto capítulo é descrita a especificação do protótipo, assim como detalhes sobre sua implementação.

No quinto capítulo são apresentadas as conclusões e sugestões a respeito do trabalho desenvolvido.

2 LINGUAGEM PL/SQL E SQL

Tendo em vista que a linguagem PL/SQL se apresenta como uma linguagem procedural, bloco-estruturada contendo comandos SQL, este capítulo se encontra subdivido em duas seções: uma trata da linguagem de consulta SQL, demonstrando seu histórico e evolução, demonstrando as principais estruturas; a outra trata da linguagem de programação PL/SQL, com seu histórico, construção e estruturas.

2.1 LINGUAGEM DE CONSULTA SQL

O termo “SQL” é originado do acrônimo de “Structured Query Language”- Linguagem de Consulta Estruturada. Segundo Date (1989), a linguagem SQL é composta por um grupo de facilidades para definição, manipulação e controle de dados de um banco de dados relacional. A partir da publicação do trabalho de E.F. Codd, “A Relational Model of Data for Large Shared Data Banks” – um modelo relacional de dados para grandes bancos de dados compartilhados, em 1970, várias empresas, incentivadas por este trabalho, desenvolveram produtos para suportar a gerência de um banco de dados.

Em 1974, nos laboratórios da IBM, definida por D. D. Chamberlin e outros, surgiu a linguagem Structured Query Language (SEQUEL). No ano seguinte, foi implementado um protótipo da mesma, chamado SEQUEL-SRM. No final dos anos 70, e início dos anos 80, a linguagem SQL foi implementada em alguns produtos da própria IBM, e também de outros fabricantes, que também passaram a suportar a linguagem.

O interesse pela padronização da linguagem iniciou-se em 1982, por intermédio do órgão de padronização American National Standards (ANSI). O processo de padronização foi baseado na linguagem SQL definida pela IBM. Posteriormente, os padrões foram reconhecidos também pela International Standards Organization (ISO), sob forma da norma ISO 9075 Date (1989). A seguir são apresentados os principais elementos da linguagem SQL.

2.1.1 LITERAIS

Os termos “literal” e “valor constante” são sinônimos, e referem-se a valores de dados fixos. Como exemplo, pode-se citar ‘BCC’ e ‘101’, que são literais caractere e inclusos entre aspas simples e 500, que é um literal numérico (Oracle (1997a)).

Muitas expressões e funções do SQL necessitam de valores literais caractere e numéricos. Literais também podem ser expressões como parte de expressões e condições.

2.1.2 COMENTÁRIOS

Comentários não afetam a execução de procedimentos SQL, mas aumentam a legibilidade das aplicações, portanto, devem ser usados sempre que possível.

Comentários podem ser expressos de duas formas: linha única e linhas múltiplas.

Comentário de linha única é representado pelo delimitador “--“, conforme o quadro 1:

Quadro 1 – Comentário de linha única

```
 SELECT cd_empregado,
        nome
   FROM empresa
  WHERE cd_empresa = 65;
 -- AND cd_filial  = 134;       Esta linha não será considerada
```

Fonte: Adaptado de Oracle (1997a).

O comentário de múltiplas linhas é representado pelos delimitadores “/*” para iniciar um comentário, e “*/” para encerrar um comentário, confome o quadro 2:

Quadro 2 – Comentário de múltiplas linhas

```
/*  A rotina abaixo bloqueia todos os clientes devedores
    Implementado em 26/02/2001
    Solicitacao de servico 541212  */

UPDATE cliente
   SET id_bloqueado = ‘SIM’
 WHERE dt_liquidacao >= TRUNC(SYSDATE);
```

Fonte: Adaptado de Oracle (1997a).

2.1.3 TIPOS DE DADOS

Todo literal ou coluna manipulada pelo Sistema Gerenciador de Banco de Dados (SGBD) Oracle tem um tipo de dado, sendo que o tipo de dado do valor associa um conjunto de propriedades, que servem para que o banco de dados faça o tratamento correto para cada valor.

No processo de criação de tabelas, clusters e procedimentos armazenados no banco, são utilizados tipos de dados internos. Estes tipos de dados definem o domínio de valores que cada coluna ou argumento pode possuir.

Quadro 3 – Tipos de Dados Internos

Tipo de Dado	Descrição
VARCHAR2(tam)	String de tamanho variável em bytes com máximo de 4000 e mínimo de 1. Deve ser especificado um tamanho para VARCHAR2.
NVARCHAR2(tam)	String de tamanho variável em bytes com máximo de 4000 e mínimo de 1, de acordo com o conjunto de caracteres nacional escolhido. Deve ser especificado um tamanho para NVARCHAR2.
NUMBER(p,s)	Numérico com precisão *p* e escala *s*. A precisão pode variar de 1 a 38 e a escala de –84 até 127.
LONG	Dados em formato caracterer de tamanho 2 gigabytes, ou 2^{31} -1 bytes.
DATE	Período de data válida de 1º de janeiro de 4712 AC até 31º Dezembro de 4712 DC.
RAW(tam)	Dados binários em formato Raw com tamanho em bytes. O tamanho deve ser especificado e é no máximo 2000.
LONG RAW	Dados binários de tamanho variável até 2 gigabytes.
ROWID	String hexadecimal que representa o endereço único de uma linha em uma tabela.
CHAR(tam)	Caracter de tamanho fixo, com tamanho em bytes. Possui tamanho mínimo e padrão de 1 byte e máximo de 2000 bytes.
NCHAR(tam)	Caracter de tamanho fixo, com tamanho em bytes. Possui tamanho mínimo e padrão de 1 byte e máximo de 2000 bytes, de acordo com o conjunto de caracteres nacional escolhido.
MLSLABEL	Formato binário de um label de sistema operacional. Usado para manter compatibilidade com versões antigas.
CLOB	Objeto caracter grande contendo caracteres de byte simples. Tamanho variável de conjunto de caracteres não é permitido. Tamanho máximo de 4 gigabytes.
NCLOB	Objeto caracter grande contendo caracteres de múltiplos bytes. Tamanho máximo de 4 gigabytes. Usado para armazenar dados de conjuntos de caracteres nacionais.
BLOB	Objeto binário grande. Tamanho máximo é de 4 gigabytes.
BFILE	Contém o localizador para um arquivo binário de grande armazenado fora da base de dados. Tamanho máximo de 4 gigabytes.

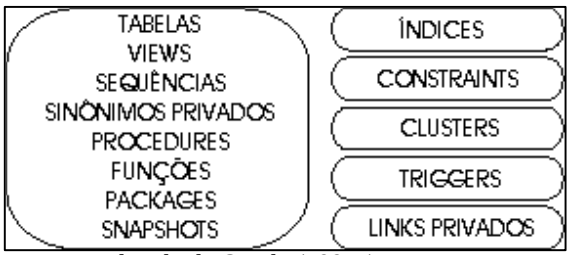
Fonte: Traduzido de Oracle (1997a).

2.1.4 NOMES DE OBJETOS DO ESQUEMA E QUALIFICADORES

Para nomear objetos no SQL, deve-se seguir as seguintes regras:

1. O tamanho dos nomes deve variar de 1 a 30, exceto nos seguintes casos:
   - Nomes de banco de dados, limitados em 8 caracteres;
   - Nomes de links de banco de dados, que podem ter até 128 caracteres;
2. Nomes não podem ter aspas simples;
3. Nomes não possuem consistência quanto à minúsculas e maiúsculas;
4. Nomes devem sempre começar com um caracter alfabético do conjunto de caracteres do banco de dados respectivo, exceto se incluso entre aspas duplas;
5. Nomes podem conter apenas caracteres alfanuméricos do conjunto de caracteres do banco de dados respectivo, e os caracteres "_" , "$" e "#". Nomes de links de banco de dados podem conter pontos e arrobas;
6. Nomes não podem usar palavras reservadas pelo Oracle. Ver anexo 2. Nomes podem ser restringidos por palavras reservadas de aplicações específicas;
7. Nome não pode ser DUAL, porque DUAL é o nome de uma tabela dummy;
8. Nomes iguais às palavras chave do banco de dados Oracle devem ser evitados por prejudicar a legibilidade da aplicação;
9. Nomes não podem ser iguais dentro de um mesmo espaço. A figura 1 demonstra os espaços de nomes em um esquema;

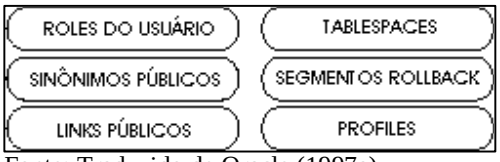
Figura 1 – Espaços de nome em um Esquema.



Fonte: Traduzido de Oracle (1997a).

- Objetos que são comuns para o banco de dados em geral devem respeitar a mesma regra. A figura 2 demonstra estes objetos.

Figura 2 – Espaços de Nomes para Objetos que não pertencem ao esquema.



Fonte: Traduzido de Oracle (1997a).

10. Colunas na mesma tabela não podem ter o mesmo nome, porém isto é permitido em tabelas diferentes;
11. Procedures e funções dentro uma package podem ter o mesmo nome, mas os argumentos devem ser diferentes;
12. Nomes podem ser colocados entre aspas duplas, contendo qualquer caractere;
13. Nomes criados com aspas duplas devem ser sempre referenciados com aspas duplas.

2.1.5 OPERADORES

Um operador manipula itens de dados e retornam um resultado. Os itens de dados são chamados de operadores ou argumentos. Operadores são representados por caracteres especiais ou palavras chaves (Oracle 1997a).

Existem duas classes de operadores:

Unário	Um operador unário opera apenas sobre um operando. Geralmente é representado neste formato: `Operador operandando`
Binário	Um operador binário opera sobre dois operandos e é representado neste formato: `Operando1 operador operando2`

Os Operadores Aritméticos são utilizados em uma expressão para negar, adicionar, subtrair, multiplicar e dividir valores numéricos. O valor da operação é numérico, e alguns destes operadores também podem ser utilizados para realizar cálculos em datas.

Quadro 4 – Operadores Aritméticos

Operador	Função
+ -	Denotam uma expressão positiva ou negativa. São operadores unários.
* /	Multiplicação, divisão. São operadores binários.
+ -	Adição, subtração. São operadores binários.

Fonte: Traduzido de Oracle (1997a).

O operador de concatenação (“ || ”) manipula strings de caracteres. Deve ser respeitado o limite de 2000 caracteres para o tipo de dado CHAR e 4000 caracteres para o tipo de dado VARCHAR.

Os operadores de comparação comparam uma expressão com outra. O resultado pode ser TRUE, FALSE ou UNKNOWN.

Quadro 5 – Operadores de Comparação

Operador	Função
=	Teste de igualdade.
!= ^= <> ¬=	Teste de diferente.
> <	Teste de “maior que” e “menor que”.
>= <=	Teste de “maior ou igual a” e “menor ou igual a”
IN	Teste de "Igual a qualquer membro de". Equivalente a "= ANY".
NOT IN	Equivalente a "!=ALL". Resulta em FALSE se qualquer membro do conjunto é NULL.
ANY SOME	Compara um valor a cada valor de uma lista retornada por uma pesquisa. Deve ser precedido por =, !=, >, <, <=, >=. Retorna FALSE se a pesquisa não retornar nenuma linha.
ALL	Compara um valor a cada valor de uma lista retornada por uma pesquisa. Deve ser precedido por =, !=, >, <, <=, >=. Retorna TRUE se a pesquisa não retornar nenhuma linha.
[NOT] BETWEEN x AND y	Maior ou igual a *x* e menor ou igual a *y*. Aceita a cláusula [NOT]
EXISTS	Retorna TRUE se uma subpesquisa retornar pelo menos uma linha.
x [NOT] LIKE y [ESCAPE 'z']	Retorna TRUE se *x* combinar com o padrão especificado em *y*, aceita a cláusula [NOT]. Qualquer caractere, exceto percentual (%) e underscore (_) pode ser usado após o ESCAPE.
IS [NOT] NULL	Teste para NULL. Ao trabalhar com nulos, este é o único operador que deve ser utilizado para testes.

Fonte: Traduzido de Oracle (1997a).

Um operador lógico combina os resultados de duas condições componentes para produzir um único resultado baseado nelas, ou para inverter o resultado de uma condição simples.

Quadro 6 – Operadores Lógicos

Operador	Função
NOT	Retorna TRUE se a condição for FALSE. Retorna FALSE se for TRUE. Se for UNKNOWN, permanece UNKNOWN.
AND	Retorna TRUE se ambos compontentes da condição são TRUE. Retorna FALSE se qualquer uma delas for FALSE. Caso contrário, retorna UNKNOWN.
OR	Retorna TRUE se qualquer uma das condições for TRUE. Só irá retornar FALSE se ambas forem FALSE Caso contrário retorna UNKNOWN.

Fonte: Traduzido de Oracle (1997a).

Operadores de conjunto combinam o resultado de duas pesquisas componentes em um único resultado. Pesquisas contendo operadores de conjunto são chamadas pesquisas compostas.

Quadro 7 – Operadores de Conjunto

Operador	Retorna
UNION	Todas as linhas selecionadas por ambas as pesquisas.
UNION ALL	Todas as linhas selecionadas por ambas as pesquisas, inclusive as duplicadas.
INTERSECT	Todas as linhas únicas selecionadas por ambas as pesquisas.
MINUS	Todas as linhas selecionadas pela primeira pesquisa mas não pela Segunda.

Fonte: Traduzido de Oracle (1997a).

Quadro 8 – Operadores Restantes

Operador	Função
(+)	Indica que a coluna que precede este indicador é a coluna que não obrigatoriamente precisa ser relacionada com a outra coluna.
PRIOR	Checa a expressão para expressão da linha pai da linha corrente em uma pesquisa hierárquica ou estruturada em árvore. Neste tipo de pequisa, este operador deve ser usado na cláusula CONNECT BY, para definir a relação entere as linhas pai e filhas.

Fonte: Traduzido de Oracle (1997a).

2.1.6 FUNÇÕES

Um função do SQL é similar a um operador na forma como manipula um item de dado e retorna um resultado. A principal diferença é a quantidade de argumentos que uma função do SQL aceita, que pode variar de zero a dois ou mais (Oracle 1997a).

Nos quadros 9, 10, 11, 12, 13 e 14, as principais funções disponíveis no SQL, separadas de acordo com os seus argumentos de entrada e respectivas saídas.

Quadro 9 – Funções Numéricas: Argumentos e retorno numéricos

Função	Utilidade
ABS(n)	Retorna o valor absoluto de *n*
ACOS(n)	Retorna o arco cosseno de *n*
ASIN(n)	Retorna o arco seno de *n*
ATAN(n)	Retorna o arco tangente de *n*
ATAN2(m,n)	Retorna o arco tangente de *m* sobre *n*
CEIL(n)	Retorna o menor inteiro maior ou igual a *n*
COS(n)	Retorna o cosseno de *n*
COSH(n)	Retorna o cosseno hiperbólico de *n*
EXP(n)	Retorna "e" elevado a *n*
FLOOR(n)	Retorna o maior inteiro menor ou igual a *n*
LN(n)	Retorna o logaritmo natural de *n*
LOG(m,n)	Retorna o logaritmo base *m* de *n*
MOD(m,n)	Retorna o resto de *m* divido por *n*
POWER(m,n)	Retorna *m* elevado a *n-ésima* potência
ROUND(n[,m])	Retorna *n* arrendondado *m* casas decimais
SIGN(n)	Retorna –1 se $n < 0$, 0 se $n = 0$ ou 1 se $n > 0$
SIN(n)	Retorna o seno de *n*
SINH(n)	Retorna o seno hiperbólico de *n*
SQRT(n)	Retorna a raiz quadrada de *n*
TAN(n)	Retorna a tangente de *n*
TANH(n)	Retorna a tangente hiperbólica de *n*
TRUNC(m,n)	Retorna *m* truncado em *n* lugares decimais

Fonte: Adaptado de Oracle (1997a).

Quadro 10 – Funções de Caracteres: retornam caracteres

Função	Utilidade
CHR(n [USING NCHAR_CS])	Retorna o caractere que possui o código binário equivalente especificado em *n*
CONCAT(char1, Char2)	Retorna *char1* concatenado com *char2*
INITCAP(char)	Retorna *char* com cada palavra com a letra inicial em maiúsculo
LOWER(char)	Retorna *char* com todas as letras em minúsculo
LPAD(char1,n,char2)	Retorna *char1*, com *n* ocorrências de *char2* à esquerda
LTRIM(char [,set])	Retorna *char*, com os caracteres de *set* à esquerda removidos
REPLACE(char1, char2,[char3])	Retorna *char1*, com todas as ocorrências de *char2* dentro de *char1* substituídas por *char3*
RPAD(char1,n,char2)	Retorna *char1*, com *n* ocorrências de *char2* à direita
RTRIM(char [,set])	Retorna *char*, com os caracteres de *set* à direita removidos
SOUNDEX(char)	Retorna a representação fonética de *char*
SUBSTR(char, m ,n)	Retorna uma parte de *char*, iniciando em *m*, com tamanho *n*
SUBSTRB(char, m ,n)	Retorna uma parte de *char*, iniciando em *m*, com tamanho *n*. Expresso em bytes
TRANSLATE(char1, char2, char3)	Retorna *char1*, com todas as ocorrências de *char2* substituídas por *char3*
UPPER(char)	Retorna *char* com todas as letras em maiúsculo

Fonte: Adaptado de Oracle (1997a).

Quadro 11 – Funções de Caracteres: retornam números

Função	Utilidade
ASCII(char)	Retorna a representação decimal de *char*
INSTR (char1,char2 ,n,m)	Retorna a *m-ésima* posição de *char2* em *char1*, iniciando a partir da posição *n*
INSTRB(char1,char2 ,n,m)	Retorna a *m-ésima* posição de *char2* em *char1*, iniciando a partir da posição *n*. Expresso em bytes
LENGTH(char)	Retorna o tamanho de *char*
LENGTHB(char)	Retorna o tamanho de *char* em bytes

Fonte: Adaptado de Oracle (1997a).

Quadro 12 – Funções de Data: retornam datas

Função	Utilidade
ADD_MONTHS(d,n)	Retorna a data *d*, adicionando-se *n* meses
LAST_DAY(d)	Retorna o último dia do mês informado em *d*
MONTHS_BETWEEN (d1, d2)	Retorna a quantidade de meses entra a data *d1* e a *d2*
NEXT_DAY(d, char)	Retorna o próximo dia da semana que corresponde a *char* após a data informada em *d*
ROUND(d[,fmt])	Retorna a data *d* arrendondada de acordo com o formato *fmt*
SYSDATE	Retorna a data e hora atual
TRUNC(d,[fmt])	Retorna *d* truncado de acordo com o formato *fmt*

Fonte: Adaptado de Oracle (1997a).

Quadro 13 – Funções de Conversão de Tipo de Dados

Função	Utilidade
CHARTOROWID(char)	Converte um valor em *char* para *rowid*
CONVERT(char, dest_char_set ,source_char_set)	Converte *char* do *source_char_set* para *dest_char_set*
HEXTORAW(char)	Converte de *hexadecimal* para *raw*
RAWTOHEX(raw)	Converte de *raw* para *hexadecimal*
ROWIDTOCHAR(rowid)	Converte de *rowid* para *char*
TO_CHAR	Converte *number* e *date* para *char*
TO_DATE	Converte *char* para *date*
TO_NUMBER	Converte *char* para *number*

Fonte: Adaptado de Oracle (1997a).

Quadro 14 – Funções de Agrupamento

Função	Utilidade
AVG(n)	Retorna o valor médio de *n*
COUNT(expr)	Retorna a quantidade de linhas retornadas por uma pesquisa, baseado em *expr*
MAX(expr)	Retorna o valor máximo de *expr*
MIN(expr)	Retorna o valor mínimo de *expr*
STDDEV(x)	Retorna o desvio padrão de *x*
SUM(n)	Retorna o somatório de *n*
VARIANCE(x)	Retorna a variância de *x*

Fonte: Adaptado de Oracle (1997a).

2.1.7 EXPRESSÕES

A expressão é a combinação de um ou mais valores, operadores e funções do SQL que resultam em um valor. Uma expressão geralmente assume o tipo de dado dos seus componentes. Abaixo, um exemplo de uma expressão, que utiliza funções e operadores; esta expressão adiciona sete dias para a data atual, remove a quantidade de horas adjacentes e converte o resultado para o tipo de dado CHAR (Oracle 1997a):

```
TO_CHAR(TRUNC(SYSDATE+7))
```

As expressões podem ser utilizadas em:

a) a lista de seleção do comando SELECT;
b) como condição das cláusulas WHERE e HAVING;
c) as cláusulas CONNECT BY, START WITH, e ORDER BY;
d) a cláusula VALUES do comando INSERT;
e) a cláusula SET do comando UPDATE.

2.1.8 CONDIÇÕES

Um condição especifica uma combinação de uma ou mais expressões e operadores lógicos que resultam em VERDADEIRO, FALSO ou desconhecido, segundo Oracle (1997a).

A condição pode ser usada na cláusula WHERE destes comandos:

a) DELETE;
b) SELECT;
c) UPDATE.

No comando SELECT, podem ser usadas condições nas seguintes cláusulas:

a) WHERE;
b) START WITH;
c) CONNECT BY;
d) HAVING.

Pode-se dizer que a condição seria um tipo de dado “lógico”, mas a linguagem não suporta formalmente este tipo.

Operadores lógicos podem combinar múltiplas condições em uma única condição, conforme o exemplo abaixo, que utiliza o operador AND para combinar duas condições:

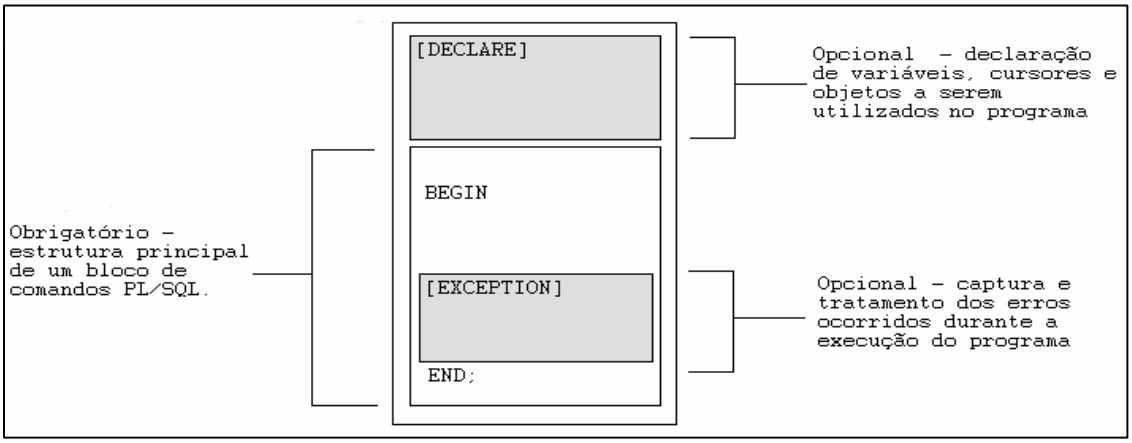
```
(1 = 1) AND (5 < 7)
```

2.2 LINGUAGEM DE PROGRAMAÇÃO PL/SQL

PL/SQL é uma linguagem de quarta geração criada pela ORACLE, como extensão procedural do SQL. Dentre as suas vantagens, pode-se destacar a integração nativa com o banco de dados e as ferramentas ORACLE, portabilidade, segurança, e algumas características de programação modernas, como encapsulamento e tratamento de exceções. Segundo Oracle (1997b), a linguagem PL/SQL combina o poder de manipulação de dados do SQL com poder de processamento de dados das linguagens procedurais. Outras linguagens que utilizam o SQL, como PRO*Cobol e Pro*C/C++ podem enviar ou executar diretamente no banco de dados segmentos de código PL/SQL.

A estrutura de um programa PL/SQL pode ser observada conforme figura 3:

FIGURA 3 - Estrutura básica de um programa PL/SQL



Fonte: Adaptado de Oracle (1997b).

2.2.1 CONJUNTO DE CARACTERES

A linguagem PL/SQL permite o seguinte conjunto de caracteres (Oracle 1997b):

a) As letras maiúsculas e minúsculas A .. Z, a .. z

b) os numéricos 0 .. 9

c) tabulações, espaçoes e saltos de linha

d) os símbolos ( ) + - * / < > = ! ~ ; : . ' @ % , " # $ ^ & _ | { } ? [ ]

Ao contrário da linguagem C, não existe consistência nesta linguagem em relação às letras maiúsculas e minúsculas, exceto quando o texto se encontra entre aspas simples.

2.2.2 UNIDADES LÉXICAS

Uma linha de código fonte PL/SQL contém um grupo de caracteres conhecidos como unidades léxicas, e podem ser classificados em:

a) delimitadores (símbolos simples e compostos)

b) identificadores, incluindo palavras reservadas

c) literais

d) comentários

2.2.3 DELIMITADORES

Um delimitador é um símbolo simples ou composto que possui um significado especial para o PL/SQL, como por exemplo os delimitadores que representam operações aritiméticas.

Quadro 15 – Símbolos Simples

+	operador de adição
%	indicador de atributo
'	delimitador de caracter ou string
.	selecionador de componente
/	operador de divisão
(	delimitador de expressão ou lista
)	delimitador de expressão ou lista
:	indicador de variável host
,	separador de item
*	operador de multiplicação
"	delimitador de identificador entre aspas
=	operador relacional
<	operador relacional
>	operador relacional
@	indicador de acesso remoto

;	terminador de procedimentos
-	operador de subtração/negação

Fonte: Traduzido de Oracle (1997b).

Quadro 16 – Símbolos Compostos

**	operador de exponenciação
<>	operador relacional
!=	operador relacional
~=	operador relacional
<=	operador relacional
>=	operador relacional
:=	operador de atribuição
=>	operador de associação
..	operador indicativo de faixa de valores
||	operador de concatenação
<<	(inicial) delimitador de label
>>	(final) delimitador de label
--	Indicador de comentário de linha única
/*	(inicial) delimitador de comentário de linhas múltiplas
*/	(final) delimitador de comentário de linhas múltiplas

Fonte Traduzido de Oracle (1997b).

2.2.4 IDENTIFICADORES

Identificadores são utilizados para nomear itens de programas e unidades, sendo eles: constantes, variáveis, exceções, cursores, váriaveis de cursor, subprogramas e packages. Cada identificador deve iniciar obrigatoriamente com uma letra, e ser seguido opcionalmente por outras letras, numerais, under scores e sinais de dólar; caracteres como hífen, barra e espaços não podem ser usados nos identificadores. Identificadores não são sensíveis ao caso, portanto podem ser escritos tanto com letras maiúsulas como minúsculas. O tamanho máximo de um identificador é de 30 caracteres.

2.2.5 PALAVRAS RESERVADAS

Os identificadores chamados de Palavras Reservadas possuem um significado sintático, e não devem ser redefinidas. No anexo 1, encontram-se as palavras reservadas utilizadas pela linguagem PL/SQL.

2.2.6 LITERAIS

Literal é númerico, caracter, string ou booleano explicíto, não representado por nenhum identificador.

2.2.7 COMENTÁRIOS

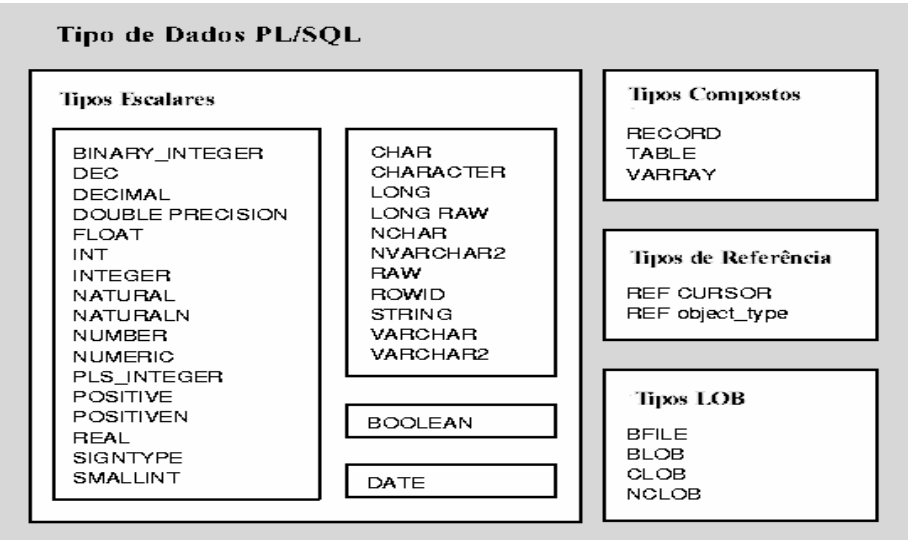
Os comentários são ignorados pelo compilador PL/SQL, mas como em qualquer linguagem, todo tipo de documentação utilizada no código fonte é útil para melhorar a legibilidade. Os comentários pode ser de dois tipos: linha única e múltiplas linhas, da mesma maneira como a linguagem SQL.

2.2.8 TIPOS DE DADOS

Toda constante e variável possui um tipo de dado, que especifica um formato de armazenamento, restrições e uma faixa de valores válida. Um tipo escalar não tem componentes internos. Um tipo composto tem componentes internos que podem ser manipulados individualmente. Um tipo de referência armazena ponteiros, que designam outros itens do programa. Um tipo Large Object (LOB) especifica a localização de objetos (imagens gráficas por exemplo), armazenadas externamente.

A figura 4 demonstra os tipos de dados disponíveis para serem utilizados em um programa PL/SQL.

FIGURA 4 – TIPOS DE DADOS DISPONÍVEIS EM UM PROGRAMA PL/SQL



Fonte: Traduzido de Oracle (1997b).

2.2.9 ESCOPO E VISIBILIDADE

Referências de um identificador são resolvidas de acordo com o seu escopo e visibilidade. O escopo de um identificador é aquela região de uma unidade de programa (bloco, subprograma ou package) da qual pode ser referenciado o identificador.

Identificadores declarados em um bloco PL/SQL são considerados locais para aquele bloco, e globais para todos os seus sub-blocos. Se um identificador é redeclarado em um sub-bloco, ambos identificadores permanecem no escopo. Dentro do sub-bloco, apenas o identificador local é visível.

2.2.10 FUNÇÕES DISPONÍVEIS INTERNAMENTE

A linguagem de programação PL/SQL possui várias funções disponíveis internamente para realizar a manipulação de dados. Essas funções podem ser divididas nas seguintes categorias:

a) informações sobre erros;
b) numéricas;
c) caractere;
d) conversão;
e) data;
f) miscelâneas.

Quadro 17 – Funções Disponíveis Internamente

Erro	Número	Caracter	Conversão	Data	Miscelânea
SQLCODE SQLERRM	ABS ACOS ASIN ATAN ATAN2 CEIL COS COSH EXP FLOOR LN LOG MOD POWER ROUND	ASCII CHR CONCAT INITCAP INSTR INSTRB LENGTH LENGTHB LOWER LPAD LTRIM NLS_INITCAP NLS_LOWER NLS_UPPER NLSSORT	CHARTOROWID CONVERT HEXTORAW NLS_CHARSET_ID NLS_CHARSET_NAME RAWTOHEX ROWIDTOCHAR TO_CHAR TO_DATE TO_LABEL TO_MULTI_BYTE TO_NUMBER TO_SINGLE_BYTE	ADD_MONTHS LAST_DAY MONTHS_BETWEEN NEW_TIME NEXT_DAY ROUND SYSDATE TRUNC	DECODE DUMP GREATEST GREATEST_LB LEAST LEAST_UB NVL UID USER USERENV VSIZE

	SIGN SIN SINH SQRT TAN TANH TRUNC	REPLACE RPAD RTRIM SOUNDEX SUBSTR SUBSTRB TRANSLATE UPPER			

Fonte: Oracle (1997b).

2.2.11 CONTROLE CONDICIONAL

As estruturas “IF” permitem executar uma sequência de comandos condicionalmente. Existem três formas de estruturas “IF”: IF-THEN, IF-THEN-ELSE e IF-THEN-ELSIF (Oracle 1997b):

Quadro 18 – Comando IF-THEN

```
IF condição THEN
   sequência_de_procedimentos;
END IF;
```

Fonte: Adaptado de Oracle (1997b).

A sequência de procedimentos é executada apenas quando a condição for TRUE. Se a condição for FALSE ou NULL, nada irá acontecer, e o controle do programa seguirá para a próxima instrução.

Quadro 19 – Comando IF-THEN-ELSE

```
IF condição THEN
   sequência_de_procedimentos1;
ELSE
   sequência_de_procedimentos2;
END IF;
```

Fonte: Adaptado de Oracle (1997b).

A sequência de procedimentos após a cláusula ELSE é executada apenas quando a condição for FALSE ou NULL.

Quadro 20 – Comando IF-THEN-ELSIF

```
IF condição1 THEN
   sequência_de_procedimentos1;
ELSIF condição2 THEN
   sequência_de_procedimentos2;
ELSE
   sequência_de_procedimentos3;
END IF;
```

Fonte: Adaptado de Oracle (1997b).

Se a primeira condição for FALSE ou NULL, a cláusula ELSIF testa outra condição. Um comando IF pode ter todos as cláusulas ELSIF necessárias, e o ELSE é opcional. As condições são testadas a partir da primeira; quando é encontrada uma condição que retorne TRUE, os procedimentos são executados, e o programa passa para o próximo comando. Se todas as condições retornarem FALSE ou NULL, serão executados os procedimentos após a cláusula ELSE, se houver.

2.2.12 CONTROLE ITERATIVO

A estrutura “LOOP” permite executar uma série de procedimentos múltiplas vezes. Existem três formas de estrutura “LOOP”:

Quadro 21 – Comando LOOP

```
LOOP
   sequência_de_procedimentos;
END LOOP;
```

Fonte: Adaptado de Oracle (1997b).

Na forma apresentada anteriormente, é apresentado um LOOP infinito; a sequência de procedimentos será executada, e ao final, a execução retorna para o topo do LOOP. Se o processamento deve ser interrompido por algum motivo, pode-se utilizar as cláusulas “EXIT”: EXIT e EXIT WHEN.

A cláusula EXIT interrompe imediatamente o processamento do “LOOP”, e passa o controle de execução para o procedimento seguinte após o “LOOP”. A cláusula EXIT WHEN é condicional, ou seja, termina o “LOOP” apenas quando a condição especificada é TRUE.

Os dois exemplos de código no quadro 22 possuem a mesma funcionalidade, mas utilizando-se a cláusula EXIT WHEN, o código se torna mais limpo e compreensível.

Quadro 22 – Cláusula EXIT WHEN

```
IF contador > 100 THEN     |     EXIT WHEN contador > 100;
   EXIT;                   |
END IF;                    |
```

Fonte: Adaptado de Oracle (1997b).

Adicionalmente, pode-se incluir também LOOP labels, assim como nos blocos PL/SQL. Desta forma, melhora-se a legibilidade do código fonte.

Quadro 23 – LOOP Label

```
<<nome_label>>
LOOP
   sequência_de_procedimentos;
END LOOP <<nome_label>>;
```

Fonte: Adaptado de Oracle (1997b).

Quadro 24 – WHILE LOOP

```
WHILE condição LOOP
   sequência_de_procedimentos;
END LOOP;
```

Fonte: Adaptado de Oracle (1997b).

Nesta forma do comando LOOP, a cada vez que o LOOP é repetido, a condição especificada no início é testada. Se em algum momento a condição for FALSE ou NULL, o LOOP irá encerrar, e o controle passará para o comando seguinte.

Quadro 25 – FOR LOOP

```
FOR contador IN [REVERSE] inteiro_inicial..inteiro_final LOOP
   sequência_de_procedimentos;
END LOOP;
```

Fonte: Adaptado de Oracle (1997b).

Nesta forma, a sequência de procedimentos irá executar de acordo com a faixa de valores definida entre “inteiro_inicial” e “inteiro_final”. Opcionalmente, pode-se utilizar a cláusula REVERSE, para que a faixa de valores execute inversamente.

2.2.13 CONTROLE SEQUENCIAL

O comando GOTO, pode ser substituído em alguns casos, por uma chamada de exceção. Assim como em outras linguagens, o uso do GOTO deve ser ponderado, visto que o excesso de chamadas deste comando pode tornar o código fonte complexo para entendimento e manutenção.

Quadro 26 – Comando GOTO

```
BEGIN
   ...
   GOTO insere_registro;
   ...
   <<insere_registro>>
   INSERT INTO emp VALUES ...
END;
```

Fonte: Adaptado de Oracle (1997b).

O seu uso é restrito dentro do próprio programa PL/SQL; não pode ser utilizado, por exemplo, para executar procedimentos dentro de um comando IF THEN, quando a chamada é executada fora deste comando, conforme quadro 27:

Quadro 27 – Comando GOTO inválido

```
BEGIN
   ...
   GOTO insere_registro;    -- esta chamada é ilegal
   ...
   IF condição THEN

      <<insere_registro>>
      INSERT INTO emp VALUES ...
   END IF;

END;
```

Fonte: Adaptado de Oracle (1997b).

O comando NULL explicitamente especifica inatividade; ele não executa nenhum tipo de ação, além de passar o controle de execução para o comando seguinte. Porém, o seu uso pode melhorar a legibilidade em algumas situações, como por exemplo, em situações de múltiplas alternativas, onde o comando NULL pode ser utilizado para demonstrar alguma situação onde nenhuma ação ou procedimento é necessária.

Quadro 28 – Comando NULL

```
BEGIN
.....

EXCEPTION
   WHEN ZERO_DIVIDE THEN
      ROLLBACK;
   WHEN VALUE_ERROR THEN
      INSERT INTO errors VALUES ...
      COMMIT;
   WHEN OTHERS THEN
      NULL;
END;
```

Fonte: Adaptado de Oracle (1997b).

3 QUALIDADE DE SOFTWARE

Este capítulo trata sobre a qualidade de software e sua importância no desenvolvimento e manutenção de programas, a influência da legibilidade e da reestruturação de software.

Segundo Bloor (1998), existem algumas atividades que são raramente ou nunca executadas pela falta de uma ferramenta que possa realizar mudanças significativas em grandes sistemas, sem um custo exorbitante. A legibilidade, e consequentemente o processo de manutenção, por exemplo, tende a se degenerar com o tempo, à medida que mudanças são realizadas no sistema. Um tempo de manutenção similar, ainda segundo Bloor (1998), poderia ser salvo, na utilização de ferramentas que redocumentem aplicações e sistemas, para mantê-los atualizados com o seu estado atual, e torná-los mais fáceis para serem compreendidos.

Um dos principais requisitos para a qualidade de um programa é a facilidade de leitura e entendimento dos procedimentos para quais ele foi criado. Segundo Arthur (1994), quando a qualidade do software melhora, os custos diminuem, os programas se tornam mais fáceis de manter e a satisfação dos usuários dispara.

Se este projeto de qualidade não for buscado durante o processo de desenvolvimento de software, a produtividade será reduzida, será encarecido o processo de desenvolvimento, e por consequência, teremos um aumento nos custos de manutenção. Por outro lado, segundo Staa (2000), a existência de critérios de qualidade bem definidos e divulgados induz os desenvolvedores a procurar satisfazê-los já no momento do desenvolvimento. Ainda segundo Staa (2000), os produtos sendo desenvolvidos possuirão uma qualidade inicial elevada, o que facilitará garantir um nível de qualidade satisfatório para terminar o desenvolvimento.

3.1 LEGIBILIDADE

Segundo Parikh (1990), as técnicas de códificação estruturada requerem que os programas sejam sistematicamente indentados para enfatizar as relações entre os segmentos de código. Esse arranjo hierárquico permite um entendimento rápido da estrutura global e local do código. Ainda segundo Parikh (1990), o poder de percepção visual pode ser derivado

desenvolvendo-se padrões sempre que for viável. Simetria, falta de simetria, indentação de bloco, regularidade, padrões de recorrência, itens similares alinhados são características que podem ser percebidas facilmente.

Pode-se afirmar que a legibilidade permite que o software seja compreendido por pessoas qualificadas, além da própria pessoa que escreveu o código.

Em um estudo apresentado por NRC (2000), realizado no Laboratório de Engenharia de Software Goddard da National Aeronautics & Space Administration (NASA), descobriu-se que a leitura manual de código-fonte é mais eficiente que testes estruturais ou funcionais para encotrar erros de codificação. A legibilidade também melhora a leitura para identificar quais partes do código-fonte serão alterados durante uma manutenção corretiva um adaptiva, e reduz a probabilidade incluir novas falhas durante este tipo de manutenção.

3.2 REESTRUTURAÇÃO

Partindo-se do princípio que a manutenção em alguns programas críticos atualmente em uso consomem uma grande quantidade de tempo dos recursos de informática de qualquer empresa, e que estes programas já foram corrigidos, alterados e documentados por várias pessoas diferentes, é necessário avaliar a possibilidade de se realizar um processo de reestruturação no código-fonte, utilizando-se padrões, a fim de deixar o código familiar para qualquer pessoa qualificada.

Segundo Arthur (1994), a evolução perfeccionista do software incluir reestruturar os códigos mediocremente escritos para torná-los mais fáceis de manter, normalizar e reestruturar bancos de dados para simplificar e aumentar seu potencial de reutilização; e revisar ou reescrever a documentação para melhorar a facilidade de acesso e utilizabilidade. Assim, seguindo a reestruturação, melhora-se a mantenabilidade, a flexibilidade e a legibilidade do código, reduzindo o custo de correções e melhoras no software.

A reestruturação é o conjunto de esforços tomados para melhorar o software, dados e documentação, sem o objetivo de corrigir defeitos ou alterar a funcionalidade, focando apenas no aprimoramento do mesmo.

Segundo Furlan (1994), as ferramentas para reestruturação têm sido empregadas com sucesso para a melhoria da produtividade no processo de manutenção, redução de defeitos (bugs), realocação de pessoal técnico e posicionamento do código para análises subseqüentes.

3.3 PADRONIZAÇÃO

Neste tópico serão apresentados sugestões de padrões e recomendações a serem seguidas, a fim de alcançar o objetivo de criar um código-fonte que atenda as características de qualidade desejadas. Estas sugestões de padronização foram obtidas a partir da pesquisa em Oracle (1997b), NRC (2000), Staa (2000) e University of Notre Dame (1996).

Os padrões e sugestões descritos a seguir se aplicam na maioria das linguagens de programação bloco-estruturada, segundo NRC (2000) e Staa (2000):

3.3.1 INDENTAÇÃO

Uma indentação apropriada facilita a identificação de declarações, fluxos de controle, comentários não-executáveis e outros componentes do código-fonte. Indentação, por sugestão, deve ser aplicada nos seguintes pontos:

a) Blocos de código, mantidos entre cláusulas *begin* e *end*;
b) Comentários;
c) Comandos estruturados como *if..then..end if*, *loop..end loop*;
d) Quando houver níveis mais internos dentro de um mesmo bloco;
e) Declaração de variáveis e subrotinas;
f) Tratamento e chamadas de exceção.

Segundo Staa (2000), o ideal é a utilização de três a cinco espaços a cada nível de indentação. A cada subrotina, é sugerido manter uma linha branca antes e após a sua especificação, a fim de manter a clareza. Procurar utilizar um comando por linha.

3.3.2 IDENTIFICADORES

Nomes de váriaveis, funções, procedures, constantes, exceções, objetos, métodos, labels e outros identificadores que não possam ser facilmente compreendidos podem causar demora ou complicações no processo de leitura e manutenção de código-fonte, segundo NRC (2000). Algumas regras, ou sugestões, para nomear esses identificadores são de que os mesmos devem ser descritivos, consistentes, e em alguns casos, identificável por outras ferramentas de diagnóstico.

Deve-se evitar o uso do mesmo nome de identificadores para objetivos diferentes. Se for necessário o seu uso, o mesmo deve ser explicitamente comentado e documentado.

O uso de palavras-chave ou reservadas para nomear identificadores também devem ser totalmente evitado para estas estruturas.

3.3.3 COMENTÁRIOS/DOCUMENTAÇÃO

Comentários incompletos e não atualizados constantemente de acordo com o estado atual do código podem trazer problemas para a leitura, e também para a segurança do programa em geral, se for tomada algum tipo de decisão baseada nestes comentários ou documentação. Estes problemas podem ser minimizados padronizando-se o processo de codificação de um projeto ou de toda a organização.

Itens que por sugestão deveriam estar documentados no início do programa são:

a) O objetivo do programa, e como é arquivada;
b) Requerimentos de funções e performance, interfaces externas que o programa ajude a implementar;
c) Outros programas ou subprogramas chamados e suas depedências;
d) Uso de váriaveis globais e locais;
e) Seção ou departamento responsável pela sua codificação;
f) Data de criação do programa;
g) Data da última revisão, número de revisão, identificador do problema, e um breve comentário;
h) Comportamento e ações a serem tomados no caso de falha de execução;
i) Entradas e saídas, incluindo arquivos, se existirem.

3.3.4 TAMANHO PROGRAMAS E SUBROTINAS

A preocupação com o tamanho dos programas, segundo NRC (2000), foi um dos principais motivadores da programas estruturada.

Programas pequenos são muito mais simples para se manter e revisar que programas compostos de longas listagens de código. Não existe um critério específico para tamanho máximo de subprograma, tendo em vista que o mesmo pode variar para cada linguagem ou forma de programação. Um número deve ser definido de acordo com a estrutura atual de programação disponível.

3.3.5 EVITAR O USO DE LITERAIS

O uso explicíto de literais (ex.: números ou nomes fixos) dentro de um código-fonte também apresenta um sério problema, pois a legibilidade e o processo de manutenção são seriamente prejudicados, principalmente se estas declarações são utilizadas numa passagem de parâmetros ou como fator de conversão.

É mais simples atualizar apenas um valor em uma tabela ou arquivo, de acordo com a implementação, do que garantir que todos os programas que usam esta informação de forma fixa foram alterados corretamente.

3.3.6 VARIÁVEIS GLOBAIS

A legibilidade aumenta se as variáveis forem declaradas e utilizadas apenas dentro da mesma rotina, segundo NRC (2000). Estas informações podem ser disponibilizadas para outras rotinas através de interfaces controladas, que minimizam a possibilidade de interações não esperadas ou não-intencionais.

3.3.7 COMANDO GOTO

A utilização do comando GOTO ou similar, que causa desvios de execução deve ser evitado porque prejudica o processo de leitura e manutenção no código-fonte em questão. Se houver uma real necessidade do seu uso, o mesmo deve ser fartamente documentado para minimizar os problemas.

3.3.8 SUBROTINAS

No início da chamada de cada subrotina, incluir um comentário dizendo por quais programas e/ou subrotinas do mesmo programa que utilizam esse segumento de código, a fim de alertar o indíviduo que está realizado a alteração sobre os possíveis efeitos de uma alteração em uma rotina acessada por outras.

3.3.9 PADRÕES E SUGESTÕES PL/SQL

As sugestões de padronização a seguir referem-se exclusivamente para a linguagem PL/SQL e foram fundamentadas segundo Oracle (1997b) e University of Notre Dame (1996):

a) Evitar o uso de if subseqüentes, prefira utilizar elsif;

Evitar:

Quadro 29 – IF Subseqüentes

```
IF condicao1 THEN
   ...
ELSE
   IF condicao2 THEN
      ...
   ELSE
      IF condicao3 THEN
         ...
      END IF;
   END IF;
END IF;
```

Fonte: Adaptado de Oracle (1997b).

Utilizar:

Quadro 30 – Comando ELSIF

```
   IF condicao1 THEN
      ...
   ELSIF condicao2 THEN
      ...
   ELSIF condicao3 THEN
      ...
   END IF;
```

Fonte: Adaptado de Oracle (1997b).

b) Usar sempre um label para identificar o loop, para assim usá-lo na hora de fechar o mesmo, melhorando a legibilidade;

Exemplo:

Quadro 31 – LOOP Label

```
<<processamento>>
LOOP

   sequencia_de_comandos;

END LOOP processamento;
```

Fonte: Adaptado de Oracle (1997b).

c) Nunca referenciar uma variável sem atribuir um valor anteriormente ou inicializá-la;

d) Escrever sempre palavras-chave e reservadas em maiúsculo;

e) Escrever sempre colunas de tabelas, variáveis, identificadores, parâmetros e alias de tabelas em minúsculo;

Exemplo:

Quadro 32 – Formatação de variáveis e identificadores

```
SELECT tab1.cd_empresa,
       tab2.cd_empregado
  FROM empregado   tab2,
       empresa     tab1
 WHERE tab1.cd_empresa = tab2.cd_empresa;
```

Fonte: Adaptado de University of Notre Dame (1996).

f) Utilizar parênteses para melhorar visualização sempre que necessário;

g) Utilizar sempre o alias nos comandos SELECT, mesmo quando houver apenas uma tabela;

h) Usar o operador != ao invés de <> para facilitar a manutenção;

i) Usar sempre conversão explícita dos dados utilizando-se das funções disponíveis para o mesmo, evitando de deixar essa tarefa por conta do banco de dados. Utilizar as funções TO_CHAR, TO_NUMBER e similares para realizar a conversão explicitamente.

j) Na cláusula ORDER BY de um SELECT, usar sempre as colunas da tabela que devem ser ordenadas, e não os números; em comandos SELECT com uma grande quantidade de colunas selecionadas, a utilização de números na cláusula ORDER BY pode dificultar a manutenção.

Quadro 33 – Formatação Cláusula ORDER BY

Evitar:

```
SELECT tab1.cd_empresa,
       tab2.cd_empregado
  FROM empregado   tab2,
       empresa     tab1
 WHERE tab1.cd_empresa = tab2.cd_empresa
 ORDER BY 1,2;
```

Utilizar:

```
SELECT tab1.cd_empresa,
       tab2.cd_empregado
  FROM empregado   tab2,
       empresa     tab1
 WHERE tab1.cd_empresa = tab2.cd_empresa
 ORDER BY tab1.cd_empresa,
          tab2.cd_empregado;
```

Fonte: Adaptado de University of Notre Dame (1996).

k) Usar a função NVL() sempre for necessário para evitar o processamento e tratamento de nulos;

l) Usar sempre a cláusula %TYPE na declaração de variáveis, evitando problemas de variáveis inválidas quando é realizada alguma alteração na base de dados;

m) Declarar e utilizar cursores para comandos SELECT que irão retornar mais que 1 linha;

n) Armazenar procedures no banco de dados quando forem usadas por muitos programas;

o) Procedures e funções com similaridades devem ser agrupadas em packages;

p) Nomear as funções e as procedures de uma package de acordo com a função da mesma;

q) Utilizar o comando FOR..LOOP quando trabalhar com cursores;

r) Incluir tratamento de exceção em todo bloco PL/SQL;

s) Utilizar como sugestão, um comentário inicial antes de cada subprograma, conforme exemplo o quadro 34:

Quadro 34 – Documentação Inicial Programa

```
/*      Nome objeto              */
/*      Nome autor               */
/*      Data criacao             */
/*      Objetivo                 */
/*      Historico Manutencao     */
/*      Data            Nome     Comentario          */
/*      05/19/1995      WJG      incluir coluna na   */
/*                               tabela emp */
```

Fonte: Adaptado de University of Notre Dame (1996).

t) Utilizar como sugestão o quadro 35 para nomenclatura dos objetos utilizados em um programa PL/SQL:

Quadro 35 – Sugestão de Nomenclatura

Identificador PL/SQL	**Nomenclatura**	**Exemplo**
Tabela PL/SQL	*Nome*_table	emp_table
Registro PL/SQL	*Nome*_record	emp_record
Variável Local	V_*nome*	v_percentual_credito
Variável de Índice	i_*nome*	i_contador
Variável Global	G_*nome*	g_salario_total
Constante	*Nome*_constant	matricula_constant
Cursor	c_*nome*	c_emp
Exceção	e_*nome*	e_produto_nao_encontrado
Label	<<*nome*>>	<<principal>>
Nome de Procedure/Função	*NOME*	CALC_BONUS
Parâmetro de Cursor	B_*nome*	b_salario

Fonte: Traduzido de University of Notre Dame (1996).

4 DESENVOLVIMENTO DO SOFTWARE

4.1 REQUISITOS PRINCIPAIS DO PROBLEMA

O software irá realizar dois tipos de funções: modificar o código fonte para atender alguns dos padrões aqui sugeridos e gerar avisos a respeito de certas construções contidas no código fonte. Não será realizada nenhuma modificação na sua funcionalidade e no modo de execução.

Modificações realizadas no fonte:

a) Gerar documentação no início do programa;

b) Gerar documentação antes de cada subrotina;

c) Formatar palavras reservadas;

d) Formatar a cláusula ORDER BY;

e) Formatar as variáveis e as colunas das tabelas;

f) Incluir tratamento de exceção nos blocos BEGIN..END;

g) Realizar a indentação do código fonte.

Avisos gerados a respeito de construções no código fonte:

h) Comando GOTO;

i) Literais.

O protótipo gera também um arquivo texto com uma matriz de referência refente às tabelas utilizadas em cada subrotina, e as operações realizadas em cada uma delas.

4.2 COMPILADORES

Para efetuar a análise do código fonte, foram utilizados alguns dos conceitos de compiladores, mais precisamente a análise léxica, e alguns pontos da análise sintática.

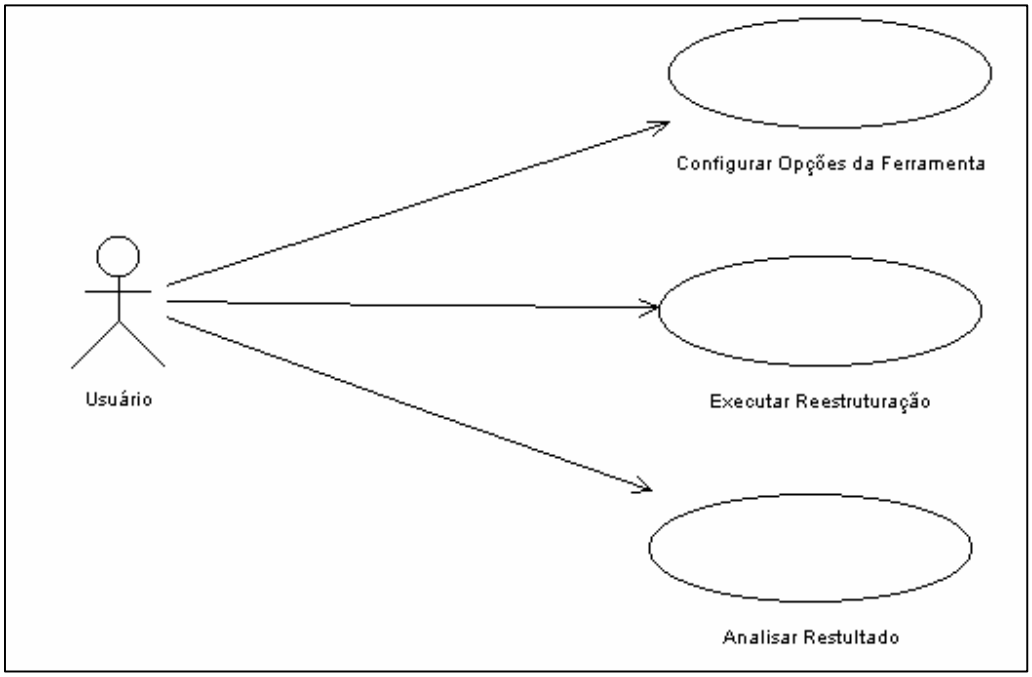
Segundo Aho (1998), o analisador léxico é a primeira fase de um compilador. Sua tarefa principal é a de ler os caracteres de entrada e produzir uma sequência de tokens para a análise sintática. Na implementação, a ferramenta lê as linhas de um arquivo texto escolhido pelo usuário, sendo o mesmo um código fonte na linguagem PL/SQL, e irá gerar uma lista em memória de todos os tokens extraídos de cada linha, sucessivamente até o final do arquivo.

O analisador sintático, segundo Aho (1998), obtém uma cadeia de tokens proveniente do analisador léxico, e verifica se a mesma pode ser gerada pela gramática da linguagem. Na implentação, a ferramenta não irá realizar uma análise sintática de todas as construções do código-fonte. É realizada uma checagem no que diz respeito a construção de certos comandos, como por exemplo, o IF THEN ELSE END IF. Segundo Zschornack (1999), na análise sintática, os tokens são agrupados em uma estrutura de árvore chamada árvore sintática que expressa a ordem em que esses elementos devem ser avaliados, e essa árvore deve expressar a precedência (ordem em que operações devem ser feitas, por exemplo, '*' antes de '+') e a associatividade (ordem em que operações devem ser feitas, considerando a mesma precedência) dos operadores de uma expressão da linguagem. Se um token não pode ser encaixado nessa árvore sintática, então ocorre um erro sintático, significando que o programa fonte está incorreto, pois não corresponde à definição da linguagem que o compilador deve reconhecer. Portanto, deve-se entender o analisador sintático como um reconhecedor de linguagens.

4.3 ESPECIFICAÇÃO

O software foi especificado usando a análise orientada a objetos, mais precisamente a Unified Modelling Language (UML). Para o diagrama de caso de uso e o diagrama de sequência foi utilizada a ferramenta CASE Rational Rose 4.0 e para especificar o diagrama de classes, foi usada a ferramenta CASE Oracle Designer 6.0, conforme explicado no capítulo 4.5.1.

FIGURA 5 – Diagrama de Caso de Uso.



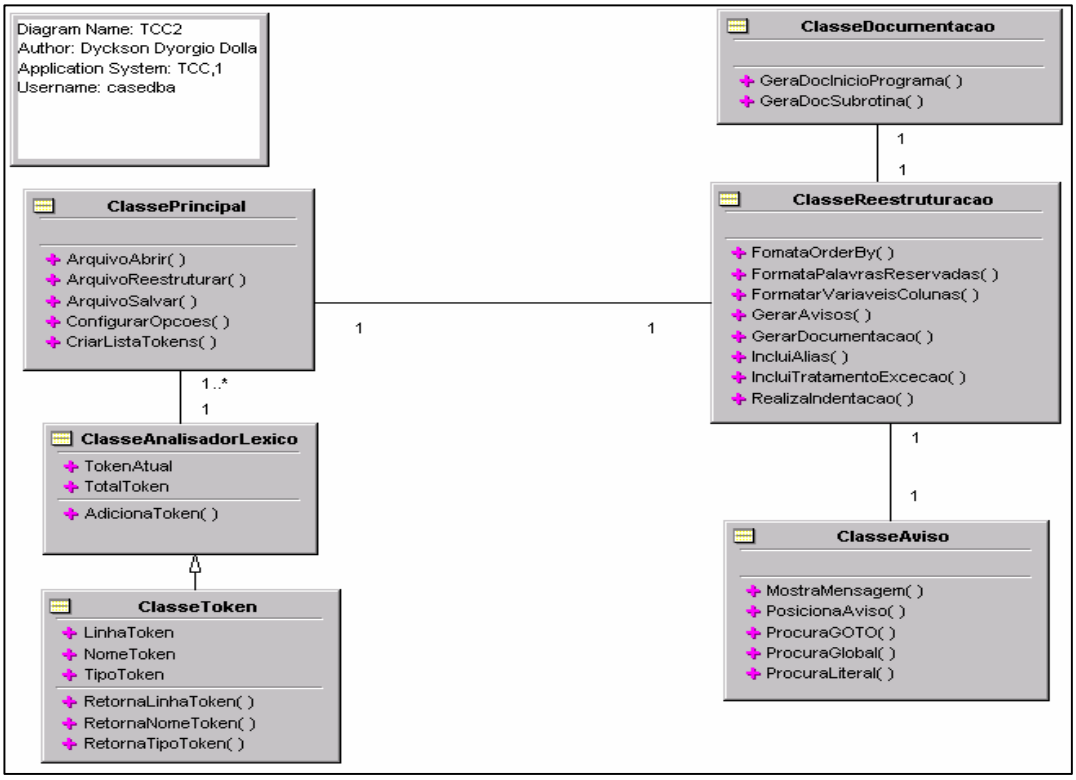
O caso de uso pode ser descrito da seguinte forma:

a) Configurar Opções da Ferramenta: o usuário antes de iniciar o uso da ferramenta deve configurar as opções da ferramenta, ou manter os padrões pré-definidos, e sugeridos neste trabalho. As opções podem ser alteradas antes de cada processo de reestruturação;

b) Executar Reestruturação: realiza o processo de reestruturação do código fonte escolhido, de acordo com as opções definidas pelo usuário na tela de configuração;

c) Analisar Resultados: usuário pode analisar o resultado do processo de reestruturação e salvar o arquivo com um novo nome, se desejado.

FIGURA 6 – Diagrama de Classes.



A seguir é apresentada uma descrição de cada classe e suas respectivas funções:

a) ClassePrincipal:

- ConfigurarOpcoes: configura opções de formatação e reestruturação da ferramenta,
- ArquivoAbrir: abre arquivo escolhido pelo usuário,
- CriarListaTokens: criar lista de tokens a partir do arquivo escolhido pelo usuário,
- ArquivoReestruturar: efetua os procedimentos de reestruturação e formatação,
- ArquivoSalvar: salva arquivo novo, reestruturado, criado a partir do arquivo antigo aberto pelo usuário;

b) ClasseAnalisadorLexico:

- AdicionaToken: adiciona à lista de tokens um novo token extraído do código fonte;

c) ClasseToken:

- RetornaLinhaToken: retorna a linha atual do token,
- RetornaNomeToken: retorna o token,
- RetornaTipoToken: retorna o tipo de token;

d) ClasseReestruturacao:

- FormataOrderBy: formata a cláusula Order By dos comandos SELECT encontrados no código fonte,
- FormataVariaveisColunas: formatar as variaveis e as colunas das tabelas de acordo com a opção escolhida pelo usuário,
- FormataPalavrasReservadas: formata as palavras reservadas encontradas de acordo com a opção escolhida pelo usuário,
- GerarAvisos: gera avisos a respeito do código fonte,
- GerarDocumentacao: gera documentação no ínicio do programa e antes de cada subrotina,
- IncluiTratamentoExcecao: incluir tratamento de exceção nos blocos BEGIN..END,
- RealizaIndentacao: realiza a indentação do código fonte de acordo;

e) ClasseDocumentacao:

- GeraDocInicioPrograma: gera a documentação no ínicio do código fonte,
- GeraDocSubrotina: gera documentação antes de cada subrotina e também uma matriz de referência das tabelas utilizadas, e os comandos SELECT, UPDATE, INSERT e DELETE realizados sobre as mesmas;

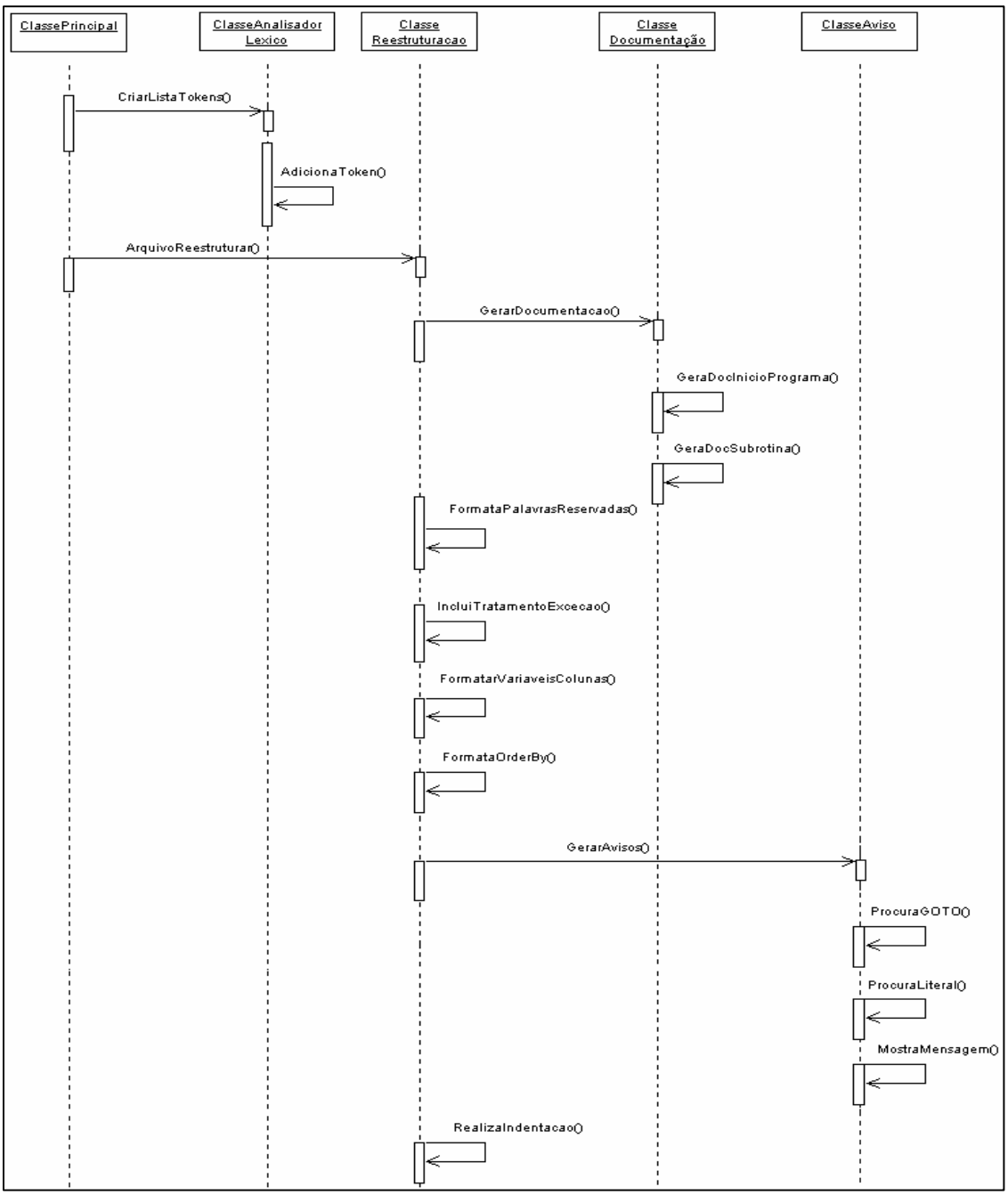
f) ClasseAviso:

- MostraMensagem: mostra as mensagens de aviso na tela do usuário,
- PosicionaAviso: posiciona o código fonte na linha escolhida pelo usuário,

- ProcuraGOTO: procura por comandos GOTO no código fonte,
- ProcuraLiteral: procurar por literais no código fonte.

Para facilitar o entendimento do Caso de Uso: Executar Reestruturação, foi elaborado o diagrama de sequência apresentado na figura 7.

FIGURA 7 – DIAGRAMA DE SEQUÊNCIA.



4.4 ESTRUTURA DO SOFTWARE

O funcionamento do software atende os seguintes passos. Primeiro é aberto um arquivo de código fonte em PL/SQL através do botão Abrir Arquivo. Depois o usuário pode, opcionalmente, escolher quais opções de reestruturação ele deseja aplicar no código-fonte. A reestruturação do código é efetuada pressionando-se o botão Executa Conversão. Após este procedimento, aparecerá na tela o código fonte reestruturado proposto.

A função de reestruturação do arquivo fonte realiza as seguintes tarefas:

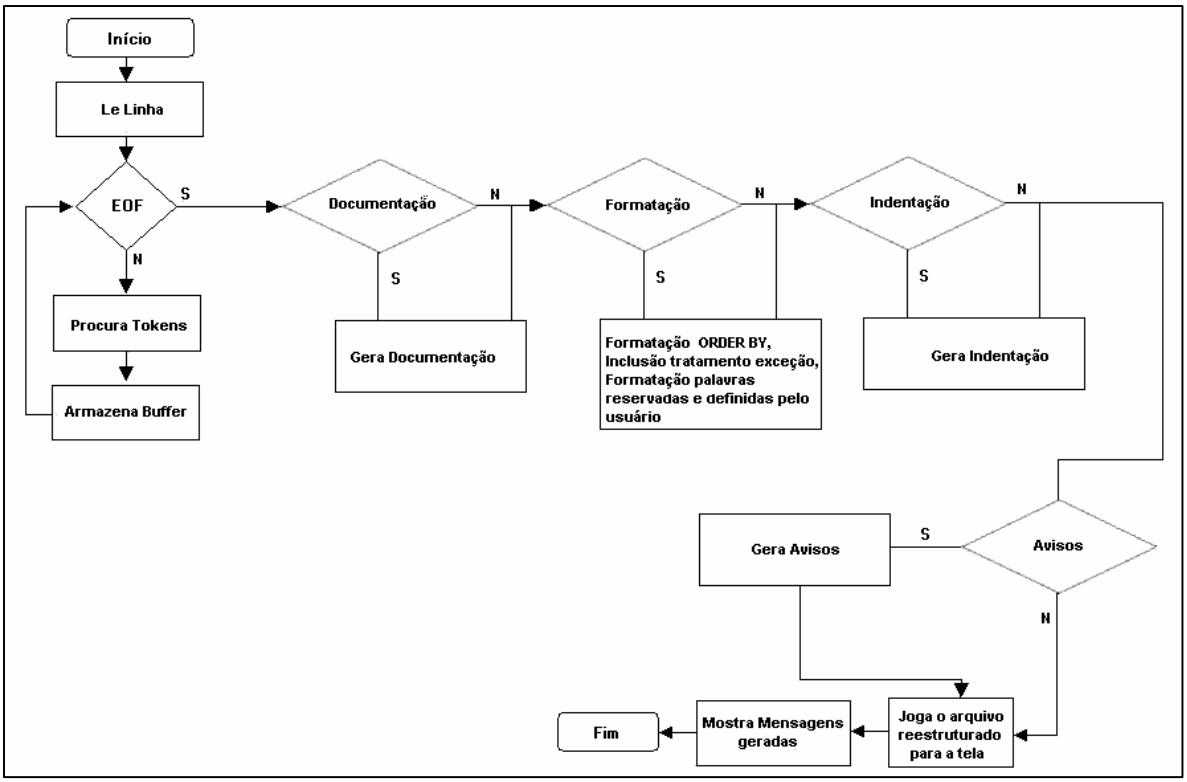
a) geração dos tokens – o arquivo é lido linha a linha, e são separados e armazenados em um buffer os tokens extraídos;

b) analisa, gera mensagens e reestrutura o arquivo aberto – o programa verifica quais pontos da reestruturação o usuário deseja executar, e efetua na seguinte ordem: documentação do programa e subrotina, indentação, formatação da cláusula ORDER BY, formatação das palavras chave/reservadas, formatação dos outros atributos da linguagem definidos pelo usuário, inclusão de tratamento de exceção, geração dos avisos e por último a indentação;

c) mostra o arquivo reestruturado e as mensagens geradas, se esta opção for escolhida – coloca o arquivo reestruturado para a tela, e mostra as mensagens e avisos gerados pela reestruturação.

A figura 8 apresenta o fluxograma de execução do programa.

FIGURA 8 – FLUXOGRAMA DE FUNCIONAMENTO DO SOFTWARE.



4.5 IMPLEMENTAÇÃO

4.5.1 TÉCNICAS E FERRAMENTAS UTILIZADAS

O prótipo foi especificado utilizando-se duas ferramentas de análise orientada a objeto: Oracle Designer versão 6.0 e o Rational Rose versão 4.0. Inicialmente seria utilizada apenas a ferramenta Oracle Designer versão 6.0, mas tendo em vista que a mesma não possuía suporte a todos os diagramas desejados, optou-se pela utilização complementar da ferramenta Rational Rose 4.0. Em ambas as ferramentas foi utilizada a notação da Unified Modelling Language (UML).

Para realizar a implementação do protótipo, a partir da especificação realizada, foi utilizada a ferramenta Borland Delphi 5. Esta ferramenta foi escolhida por possuir uma

relativa facilidade para trabalhar com arquivos texto, e algumas funções pré-definidas internamente para manipulação de strings.

Para exemplificar, foi demonstrando um exemplo do procedimento ProcuraGOTO, onde é realizado uma varredura em toda a lista de tokens, procurando pelo comando GOTO, sempre checando se o comando não está incluso dentro de um comentário interno do programa. O funcionamento do procedimento é simples: ao encontrar o comando GOTO, é gerada uma mensagem para o usuário, informando em qual linha se encontra o mesmo.

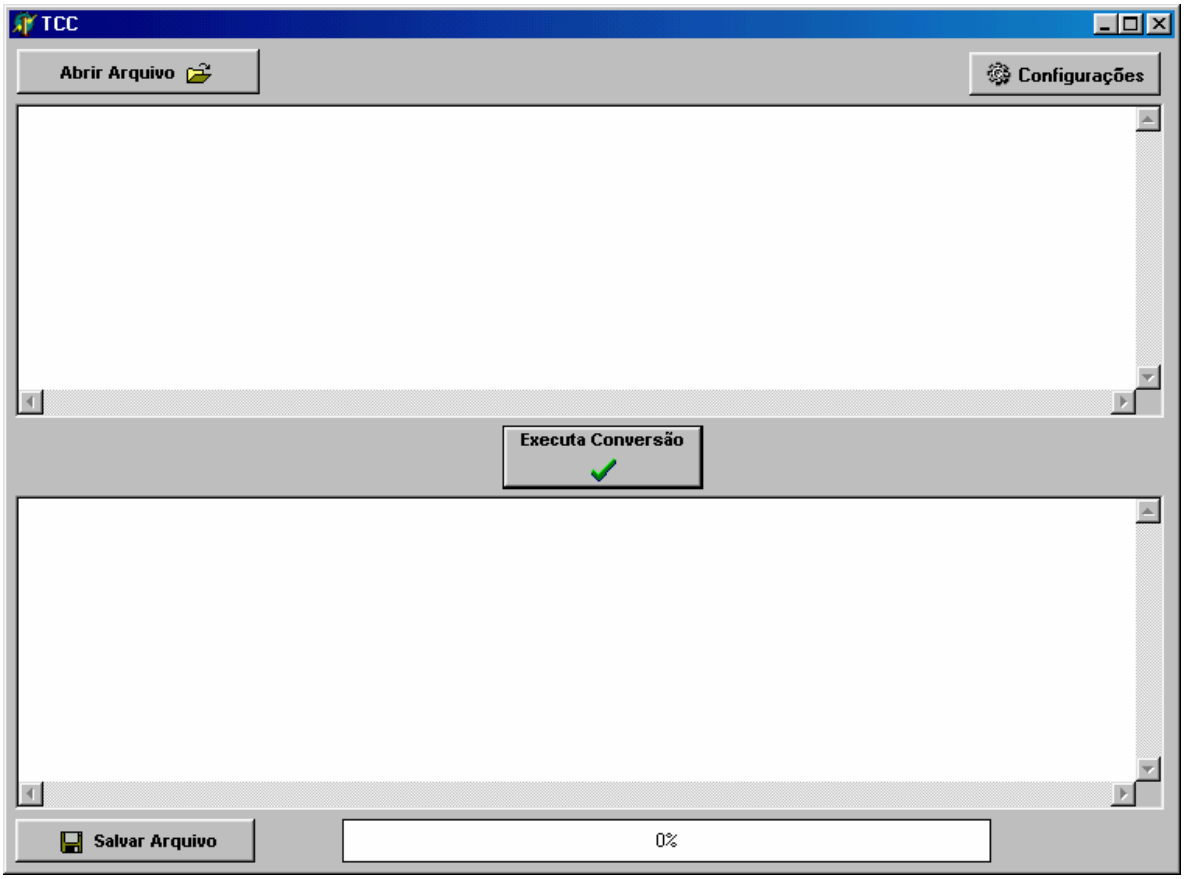
Quadro 36 – Código Fonte do Protótipo

```
procedure ProcuraGOTO;
var
   ind2,
   contador2 : integer;
begin

     contador2 := tam_array;

     for ind2:=0 to contador2 - 1 do
        begin

          if (uppercase(trimright(trimleft(ArrayOriginal[ind2].token))) = 'GOTO') and
             (ArrayOriginal[ind2].tp_token <> 'C') then
             begin
                FrmMensagem.StringMensagem.Rows[FrmMensagem.StringMensagem.Row].text :=
                           'Encontrado GOTO';
                FrmMensagem.StringMensagem.Cols[1].Append
                                          (inttostr(ArrayOriginal[ind2].num_linha));
                FrmMensagem.StringMensagem.Row := FrmMensagem.StringMensagem.Row + 1;
             end;

        end;
        FrmMensagem.StringMensagem.cursor := 1;

end;
```

4.5.2 OPERACIONALIDADE DA IMPLEMENTAÇÃO

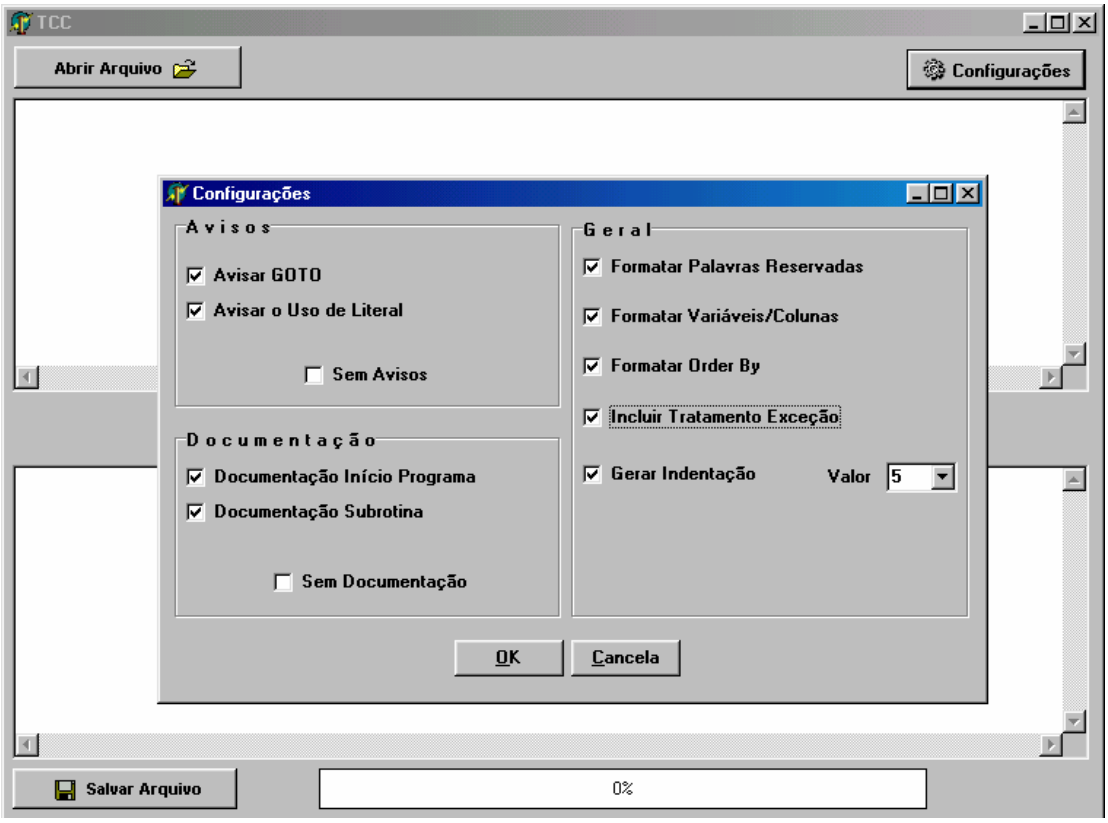
Para demonstrar a operacionalidade da implentação foi demonstrando um estudo de caso, com uma execução passo-a-passo do processo de reestruturação utilizando o protótipo construído.

FIGURA 9 – Tela principal do Protótipo.



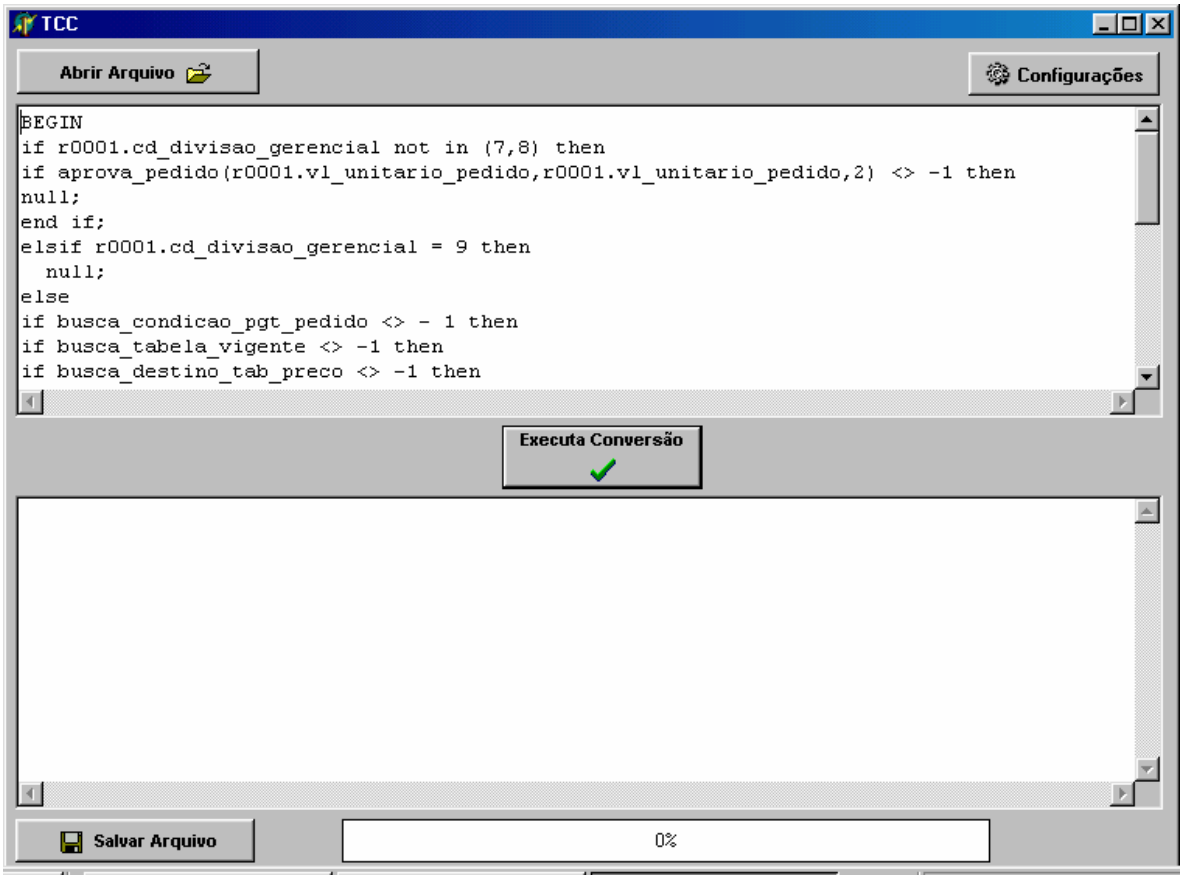
Ao executar o aplicativo, a tela inicial é apresentada, conforme a figura 9. Nesta tela o usuário pode executar, pressionando-se os botões correpondentes, as funções de escolher um arquivo para reestruturar, configurar as opções da ferramenta, efetuar o processo de reestruturação, e salvar o arquivo reestruturado.

FIGURA 10 – Tela de opções de configuração.



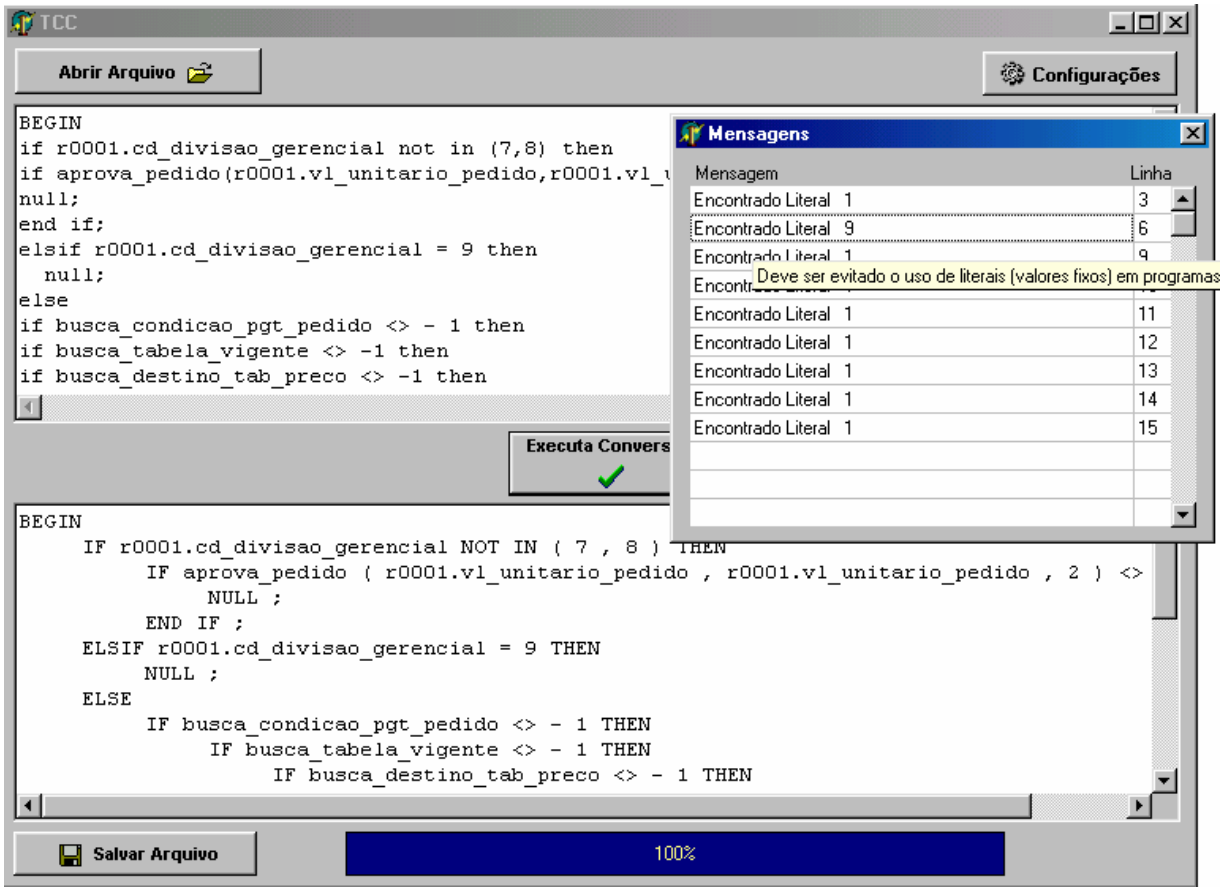
Na figura 10 pode-se observar as opções de configuração disponíveis para serem escolhidas pelo usuário. Estas opções podem ser modificadas antes ou após a escolha de um arquivo para ser reestruturado.

FIGURA 11 – Código-fonte PL/SQL sem reestruturação.



Na figura 11 tem-se um código-fonte em formato PL/SQL aberto na janela superior da ferramenta (código original). O código-fonte escolhido possui uma série de comandos *if..then..end if* subseqüentes, de díficil legibilidade.

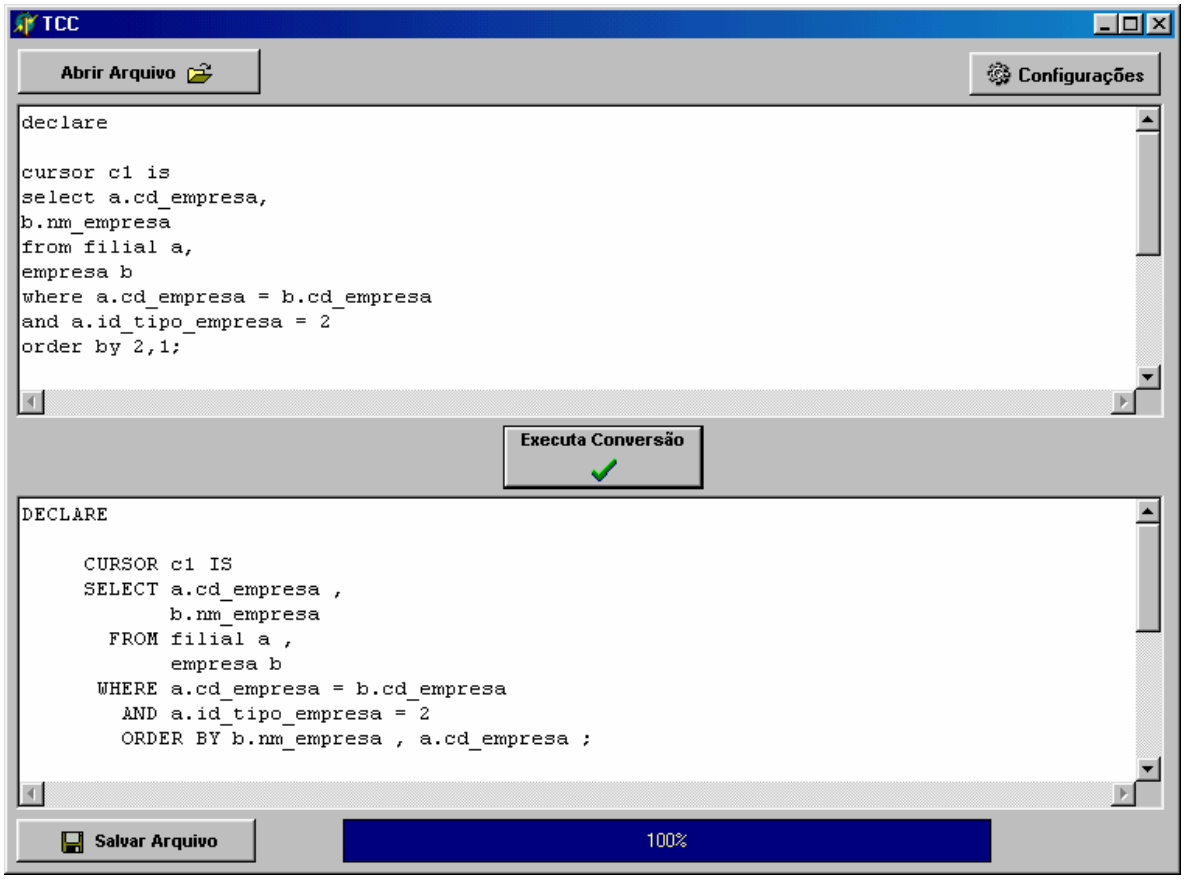
FIGURA 12 – Código-fonte PL/SQL após processo de reestruturação.



Na figura 12 tem-se o código-fonte novo, gerado a partir do processo de reestruração. Todos os comandos *if..then..end if* subseqüentes foram indentandos de acordo com o valor informado na tela de configurações. Foram formatadas também as palavras-chave ou reservadas e os nomes de váriaveis/procedimentos do programa.

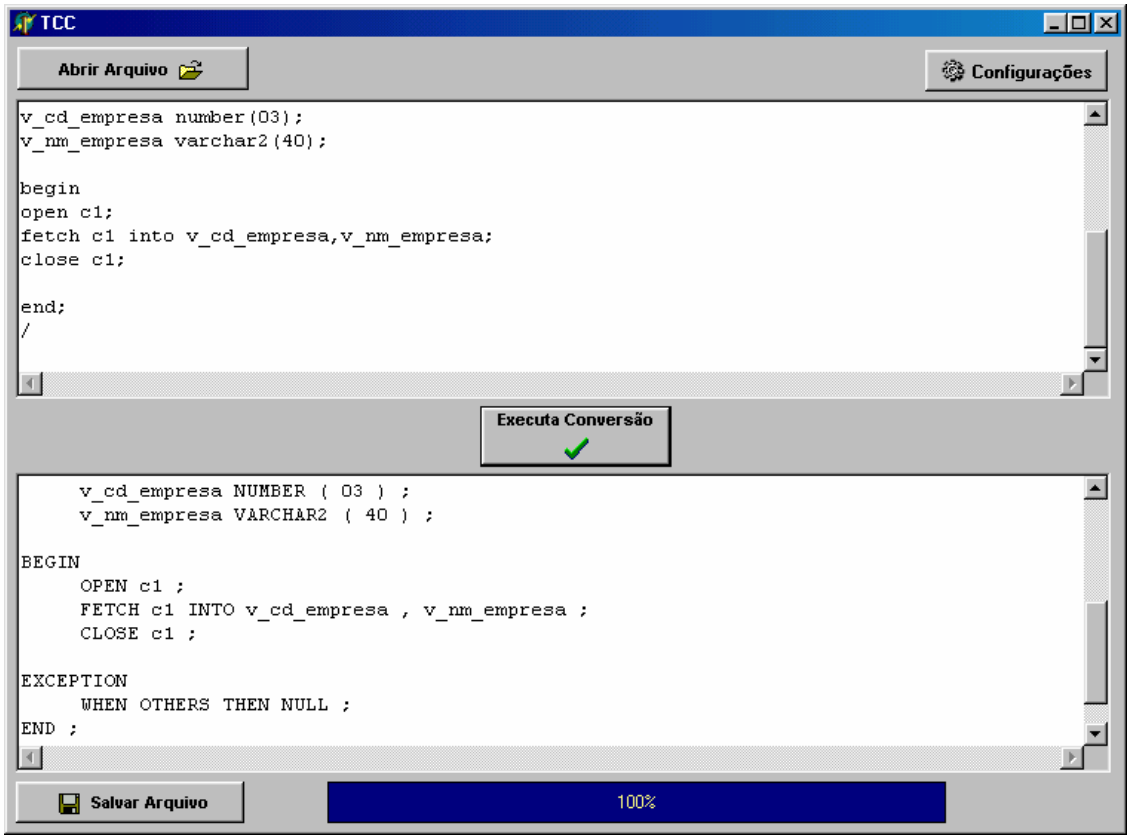
Conforme selecionado na tela de configurações, é gerada também a tela de mensagens, com avisos sobre determinadas construções encontradas no programa. Ao selecionar a linha na tela de mensagens, o usuário recebe uma breve informação do ocorrido. Com um duplo clique do mouse, o programa posiciona automaticamente em qual linha se encontra o problema.

FIGURA 13 – Exemplo de formatação de Order By e Palavras Reservadas.



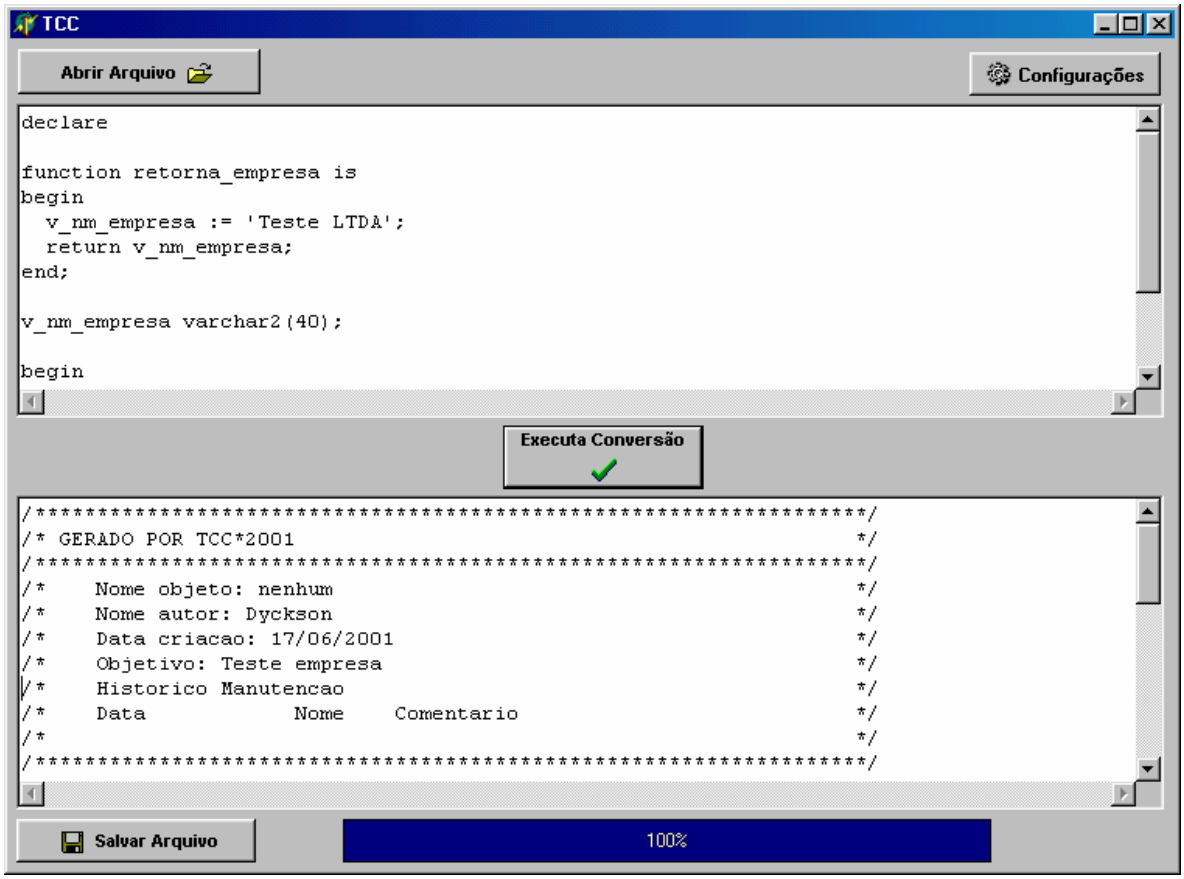
Na figura 13 tem-se a formatação das palavras reservadas/chaves e das colunas das tabelas. É demonstrada também a formatação da cláusula ORDER BY, onde os números são substituídos por suas respectivas colunas.

FIGURA 14 – Inclusão de tratamento de exceção.



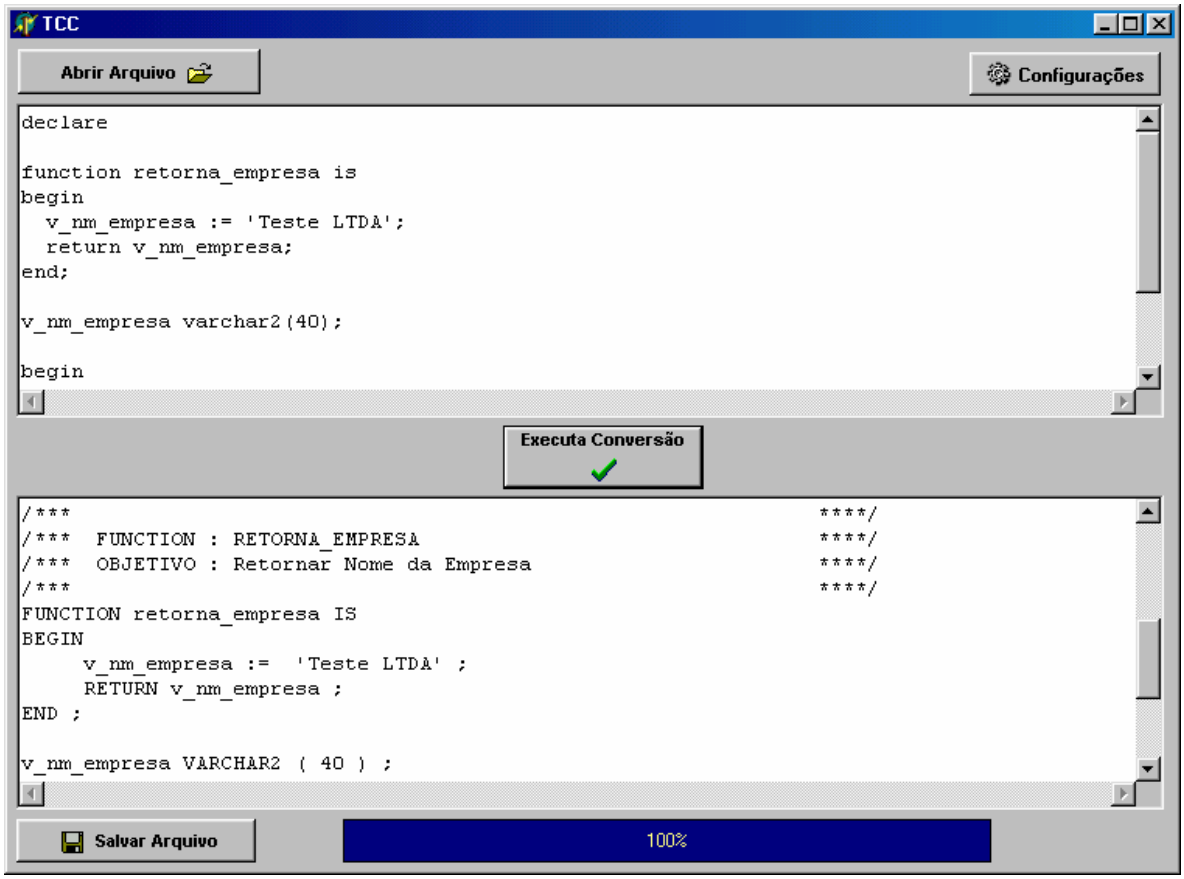
Na figura 14 além da formatação das palavras chave ou reservadas e as variáveis dos programas, é demonstrada também a inclusão do tratamento de exceção. O tratamento de exceção incluído automaticamente deve ser modificado pelo usuário para que se adapte à sua aplicação.

FIGURA 15 – Documentação no início do programa.



O protótipo gera, se o usuário escolher, uma breve documentação inicial do programa, a qual o usuário que irá efetuar as futuras manutenções pode complementar. Para gerar esta documentação, é verificada a existência do código na segunda linha (GERADO POR TCC*2001), para evitar que um código-fonte já documentado por esta ferramenta seja documentado novamente. O programa solicita que o usuário informe o nome do autor, o objetivo do programa e o nome do objeto.

FIGURA 16 – Documentação nas subrotinas do programa.



A ferramenta gera uma documentação para cada subrotina (função ou procedimento) encontrada no código-fonte. A cada subrotina encontrada, é solicitado que o usuário informe o objetivo da mesma, para que seja documentado juntamente com o programa. É gerado também um relatório do tipo matriz de referência, com as tabelas utilizadas pelas subrotinas e os comandos realizados sobre cada uma delas, conforme o exemplo da figura 17, que foi gerado a partir do terceiro código-fonte ilustrado no anexo 3.

Figura 17 – Matriz de Referência

TABELAS UTILIZADAS POR RETORNA_EMPRESA				
TABELAS	SELECT	INSERT	UPDATE	DELETE
EMPRESA	X		X	
FILIAL	X			
TABELAS UTILIZADAS POR ATUALIZA_EMPRESA				
TABELAS	SELECT	INSERT	UPDATE	DELETE
EMPRESA			X	X
FILIAL_ALTERADA		X		

4.5.3 RESULTADOS E DISCUSSÃO

A funcionalidade do protótipo atingiu a meta esperada. Foram utilizados vinte e cinco arquivos de teste com exemplos de código-fonte PL/SQL, abrangendo vários construções possíveis que fazem parte da utilização usual desta linguagem de programação. Dentre estes vinte e cinco arquivos, dez exemplos foram buscados dos próprios arquivos de configuração do banco de dados Oracle, dez exemplos foram criados manualmente a fim de satisfazer exatamente a situação que o programa iria tratar, mas sempre espelhando a realidade, e cinco exemplos são rotinas executadas em uma empresa que utiliza banco de dados Oracle no seu dia-a-dia.

Seguindo os conceitos de qualidade de software apresentados no capítulo 3, o protótipo realiza uma série de atividades de reestruturação no código-fonte, aplicando uma formatação com o objetivo de padronização e incrementando a legibilidade geral do aplicativo.

No anexo 3 foram incluídas as listagens de três programas que foram submetidos pelo processo de reestruração da ferramenta. Nos três exemplos, todas as opções da ferramenta estavam habilitadas, e o valor da indentação estava configurado para cinco espaços. Para demonstrar ainda mais a melhora na legibilidade do código-fonte com a inclusão da

indentação no processo de reestruturação, foram retirados todos os espaços iniciais das linhas dos programas exemplo constantes no anexo 3.

Nos exemplos do anexo 3, foi incluída a documentação inicial no programa, tratamento de exceção, formatação das palavras reservadas, formatação dos outros elementos da linguagem definidos pelo usuário e indentação com cinco espaços. Pode-se observar a melhora da legibilidade do código-fonte no exemplo 1 do anexo 3, o qual possui vários comandos IF..THEN END IF subseqüentes. A legibilidade neste código-fonte foi melhorada com o encadeamento correto dos comandos IF..THEN END IF.

A maneira de execução do protótipo é similar ao utilizado em Dalmolin (2000). Neste protótipo criado para a linguagem de programação PL/SQL, optou-se por parametrizar quais atributos da reestruturação o usuário deseja executar, tornando a ferramenta mais flexível.

5 CONCLUSÕES

Os objetivos do trabalho foram alcançados visto que a ferramenta permite analisar e reestruturar código-fonte de programas feitos na linguagem PL/SQL, a partir dos seguintes padrões de legibilidade estudados na bibliografia: indentação, formatação da cláusula ORDER BY, formatação das palavras chave/reservadas, formatação dos outros atributos da linguagem definidos pelo usuário, inclusão de tratamento de exceção, geração de documentação para o programa e suas subrotinas. O protótipo ainda possui, adicionalmente, a geração de avisos sobre determinadas construções da linguagem encontradas no código-fonte.

O protótipo cumpre a função de transformar um código-fonte, complicado de se manter e corrigir por falta de legibilidade, em um programa de melhor leitura e compreensão, além de estimular o processo de documentação para auxiliar as manutenções futuras em pontos-chave ou críticos do código-fonte. Foi incluído no protótipo, para aumentar a flexibilidade de execução do mesmo, a opção do próprio usuário escolher quais pontos do processo de reestruração ele deseja submeter no código-fonte.

Um processo ideal a ser seguido por todas as empresas que desenvolvem software é a adoção de um padrão no processo de criação e manutenção dos programas manufaturados pela empresa. Desta forma, se todos os recursos humanos responsáveis pela confecção do software seguirem este padrão, a qualidade alcançada pelos softwares desenvolvidos será muito superior.

As ferramentas utilizadas tanto para especificação e análise, como para programação foram adequadas para a construção deste protótipo. Restaram apenas algumas ressalvas quanto à ferramenta de programação Delphi, que requereu um profundo estudo em arquivos de ajuda, manuais e internet, para resolver algumas dificuldades encontradas durante o processo.

Não foi incluído neste trabalho o processo de reestruturação de variáveis (nomenclatura) e correções na estrutura de execução dos programas, por se tratar de um

processo muito complexo, e que muitas vezes dependem de julgamento humano. O processo de busca de literais no código-fonte também precisa ser aprimorado, pois não foram tratadas todas as suas formas de utilização.

5.1 EXTENSÕES

Uma sugestão de extensão, é a de realizar este mesmo trabalho para outras linguagens de programação disponíveis no mercado.

Com o crescimento da quantidade de pessoas que utilizam as linguagens de programação visuais (Visual Basic, Visual C++, Delphi), seria interessante criar algum tipo de padronização para as telas gráficas criadas nestes ambientes.

ANEXO 1 – PALAVRAS RESERVADAS DO PL/SQL

ABORT	DBA	INTEGER	PRIVATE	TABAUTH
ACCEPT	DEBUGOFF	INTERFACE	PRIVILEGES	TABLE
ACCESS	DEBUGON	INTERSECT	PROCEDURE	TABLES
ADD	DECLARE	INTO	PUBLIC	TASK
ALL	DECIMAL	IS	RAISE	TERMINATE
ALTER	DEFAULT	LEVEL	RANGE	THEN
AND	DEFINTION	LIKE	RAW	TO
ANY	DELAY	LIMITED	REAL	TRIGGER
ARRAY	DELETE	LOCK	RECORD	TRUE
ARRAYLEN	DESC	LONG	REF	TYPE
AS	DIGITS	LOOP	RELEASE	UID
ASC	DISPOSE	MAX	REMR	UNION
ASSERT	DISTINCT	MAXEXTENTS	RENAME	UNIQUE
ASSIGN	DO	MIN	RESOURCE	UPDATE
AT	DROP	MINUS	RETURN	USE
AUDIT	ELSE	MLSLABEL	RETURN	USER
AUTHORIZATION	ELSIF	MOD	REVERSE	VALIDATE
AVG	END	MODE	REVOKE	VALUES
BASE_TABLE	ENTRY	NATURAL	ROLLBACK	VARCHAR
BEGIN	EXCEPTION	NATURALN	ROW	VARCHAR2
BETWEEN	EXCEPTION_INIT	NEW	ROWID	VARIANCE
BINARY_INTEGER	EXCLUSIVE	NEXTVAL	ROWLABEL	VIEW
BODY	EXISTS	NOAUDIT	ROWNUM	VIEWS
BOOLEAN	EXIT	NOCOMPRESS	ROWS	WHEN
BY	FALSE	NOT	ROWTYPE	WHENEVER
CASE	FETCH	NOWAIT	RUN	WHERE
CHAR	FILE	NULL	SAVEPOINT	WHILE
CHAR_BASE	FLOAT	NUMBER	SCHEMA	WITH
CHECK	FOR	NUMBER_BASE	SELECT	WORK
CLOSE	FORM	OF	SEPERATE	WRITE
CLUSTER	FROM	OFFLINE	SESSION	XOR
CLUSTERS	FUNCTION	ON	SET	
COLAUTH	GENERIC	ONLINE	SHARE	
COLUMN	GOTO	OPEN	SIGNTYPE	
COMMENT	GRANT	OPTION	SMALLINT	
COMMIT	GROUP	OR	SPACE	
COMPRESS	HAVING	ORDER	SQL	
CONNECT	IDENTIFIED	OTHERS	SQLCODE	
CONSTANT	IF	OUT	SQLERRM	
CRASH	IMMEDIATE	PACKAGE	START	
CREATE	IN	PARTITION	STATEMENT	
CURRENT	INCREMENT	PCTFREE	STDDEV	
CURRVAL	INDEX	PLS_INTEGER	SUBTYPE	
CURSOR	INDEXES	POSITIVE	SUCCESSFUL	
DATABASE	INDICATOR	POSITIVEN	SUM	
DATA_BASE	INITIAL	PRAGMA	SYNONYM	
DATE	INSERT	PRIOR	SYSDATE	

ANEXO 2 – PALAVRAS RESERVADAS DO BANCO DE DADOS ORACLE

ACCESS	FOR*	OFFLINE	UID
ADD*	FROM*	ON*	UNION*
ALL*	GRANT*	ONLINE	UNIQUE*
ALTER*	GROUP*	OPTION*	UPDATE*
AND*	HAVING*	OR*	USER*
ANY*	IDENTIFIED	ORDER*	VALIDATE
AS*	IMMEDIATE*	PCTFREE	VALUES*
ASC*	IN*	PRIOR*	VARCHAR*
AUDIT	INCREMENT	PRIVILEGES*	VARCHAR2
BETWEEN*	INDEX	PUBLIC*	VIEW*
BY*	INITIAL	RAW	WHENEVER*
CHAR*	INSERT*	RENAME	WHERE
CHECK*	INTEGER*	RESOURCE	WITH*
CLUSTER	INTERSECT*	REVOKE*	
COLUMN	INTO*	ROW	
COMMENT	IS*	ROWID	
COMPRESS	LEVEL*	ROWNUM	
CONNECT*	LIKE*	ROWS*	
CREATE*	LOCK	SELECT*	
CURRENT*	LONG	SESSION*	
DATE*	MAXEXTENTS	SET*	
DECIMAL*	MINUS	SHARE	
DEFAULT*	MODE	SIZE*	
DELETE*	MODIFY	SMALLINT*	
DESC*	NETWORK	START	
DISTINCT*	NOAUDIT	SUCCESSFUL	
DROP*	NOCOMPRESS	SYNONYM	
ELSE*	NOT*	SYSDATE	
EXCLUSIVE	NOWAIT	TABLE*	
EXISTS	NULL*	THEN*	
FILE	NUMBER	TO*	
FLOAT*	OF*	TRIGGER	

ANEXO 3 – EXEMPLOS DE CÓDIGO-FONTE

EXEMPLO 1:

ANTES:

```
BEGIN
if r0001.cd_empresa not in (7,8) then
if aprova(r0001.vl_uni_ped,r0001.vl_uni_ped,2) <> -1 then
null;
end if;
elsif r0001.cd_empresa = 9 then
  null;
else
if condicao_pagto <> - 1 then
if tabela_vigente <> -1 then
if destino <> -1 then
if perc_icms <> -1 then
if tabela <> - 1 then
if condicoes = 1 then
if descontos <> -1 then
null;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;


END;
```

APÓS:

```
/*****************************************************************/
/* GERADO POR TCC*2001                                           */
/*****************************************************************/
/*      Nome objeto: nenhum                                       */
/*      Nome autor: Dyckson DOlla                                 */
/*      Data criacao: 20/06/2001                                  */
/*      Objetivo: teste2                                          */
/*      Historico Manutencao                                      */
/*      Data            Nome    Comentario                        */
/*                                                               */
/*****************************************************************/
BEGIN
     IF r0001.cd_empresa NOT IN ( 7 , 8 ) THEN
          IF aprova ( r0001.vl_uni_ped , r0001.vl_uni_ped , 2 ) <> - 1 THEN
               NULL ;
          END IF ;
     ELSIF r0001.cd_empresa = 9 THEN
          NULL ;
     ELSE
          IF condicao_pagto <> - 1 THEN
               IF tabela_vigente <> - 1 THEN
                    IF destino <> - 1 THEN
                         IF perc_icms <> - 1 THEN
                              IF tabela <> - 1 THEN
                                   IF condicoes = 1 THEN
                                        IF descontos <> - 1 THEN
                                             NULL ;
                                        END IF ;
```

```
                                END IF ;
                            END IF ;
                        END IF ;
                    END IF ;
                END IF ;
            END IF ;
        END IF ;
EXCEPTION
    WHEN OTHERS THEN NULL ;
END ;
```

EXEMPLO 2:

ANTES:

```
declare

cursor c1 is
select a.cd_empresa,
b.nm_empresa
from filial a,
empresa b
where a.cd_empresa = b.cd_empresa
and a.id_tipo_empresa = 2
order by 2,1;

v_cd_empresa number(03);
v_nm_empresa varchar2(40);

begin
open c1;
fetch c1 into v_cd_empresa,v_nm_empresa;
close c1;

end;
/
```

APÓS:

```
/*****************************************************************/
/* GERADO POR TCC*2001                                           */
/*****************************************************************/
/*    Nome objeto: teste                                         */
/*    Nome autor: Dyckson                                        */
/*    Data criacao: 20/06/2001                                   */
/*    Objetivo: teste                                            */
/*    Historico Manutencao                                       */
/*    Data            Nome    Comentario                         */
/*                                                               */
/*****************************************************************/
DECLARE

     CURSOR c1 IS
     SELECT a.cd_empresa ,
            b.nm_empresa
       FROM filial a ,
```

```
            empresa b
       WHERE a.cd_empresa = b.cd_empresa
         AND a.id_tipo_empresa = 2
         ORDER BY b.nm_empresa , a.cd_empresa ;

      v_cd_empresa NUMBER ( 03 ) ;
      v_nm_empresa VARCHAR2 ( 40 ) ;

BEGIN
      OPEN c1 ;
      FETCH c1 INTO v_cd_empresa , v_nm_empresa ;
      CLOSE c1 ;

EXCEPTION
      WHEN OTHERS THEN NULL ;
END ;
```

EXEMPLO 3:

ANTES:

```
declare

function retorna_empresa is
v_nm_empresa            varchar2(40);
begin

SELECT V1.NM_EMPRESA
INTO V_NM_EMPRESA
FROM EMPRESA V1,
FILIAL  V2
WHERE V2.CD_FILIAL = V2.CD_FILIAL
ORDER BY 1 DESC;


UPDATE EMPRESA
SET DT_ATUALIZACAO = SYSDATE,
ID_USUARIO = 'SFSFS'
WHERE CD_EMPRESA = 30;

return v_nm_empresa;
end;


function atualiza_empresa is
begin

DELETE FROM EMPRESA
WHERE CD_EMPRESA = 30;


UPDATE EMPRESA
SET DT_ATUALIZACAO = SYSDATE,
ID_USUARIO = 'SFSFS'
```

```
WHERE CD_EMPRESA = 30;

INSERT INTO FILIAL_ALTERADA
(CD_FIL,NM_FIL)
VALUES(V_CD_FILIAL,V_NOME_FILIAL);


return v_nm_empresa;
end;


v_nm_empresa varchar2(40);

begin

v_nm_empresa := retorna_empresa;

end;
/
```

APÓS:

```
/*****************************************************************/
/* GERADO POR TCC*2001                                           */
/*****************************************************************/
/*    Nome objeto: teste.pck                                     */
/*    Nome autor: Dyckson                                        */
/*    Data criacao: 21/07/2001                                   */
/*    Objetivo: Retornar empresa                                 */
/*    Historico Manutencao                                       */
/*    Data            Nome    Comentario                         */
/*                                                               */
/*****************************************************************/
DECLARE

/***                                                        ****/
/***  FUNCTION : RETORNA_EMPRESA                            ****/
/***  OBJETIVO : Retornar Empresa                           ****/
/***                                                        ****/
FUNCTION retorna_empresa IS
     v_nm_empresa VARCHAR2 ( 40 ) ;
BEGIN

     SELECT v1.nm_empresa
            INTO v_nm_empresa
       FROM empresa v1 ,
            filial v2
      WHERE v2.cd_filial = v2.cd_filial
        ORDER BY v1.nm_empresa DESC ;


     UPDATE empresa
     SET dt_atualizacao = SYSDATE ,
     id_usuario = 'SFSFS'
     WHERE cd_empresa = 30 ;
```

```
     RETURN v_nm_empresa ;
EXCEPTION
     WHEN OTHERS THEN raise ;
END ;


/***                                                                ****/
/***  FUNCTION : ATUALIZA_EMPRESA                                   ****/
/***  OBJETIVO : Atualizar Empresa                                  ****/
/***                                                                ****/
FUNCTION atualiza_empresa IS
BEGIN

     DELETE FROM empresa
     WHERE cd_empresa = 30 ;


     UPDATE empresa
     SET dt_atualizacao = SYSDATE ,
     id_usuario = 'SFSFS'
     WHERE cd_empresa = 30 ;

     INSERT INTO filial_alterada
     ( cd_fil , nm_fil )
     VALUES ( v_cd_filial , v_nome_filial ) ;


     RETURN v_nm_empresa ;
EXCEPTION
     WHEN OTHERS THEN RAISE ;
END ;


v_nm_empresa VARCHAR2 ( 40 ) ;

BEGIN

     v_nm_empresa :=  retorna_empresa ;

EXCEPTION
     WHEN OTHERS THEN RAISE ;
END ;
/
```

REFERÊNCIAS BIBLIOGRÁFICAS

AHO, Alfred V.; Sethi, Ravi; Ullman, Jeffrey D. **Compiladores:** princípios, técnicas e ferramentas. Rio de Janeiro:LTC – Livros Técnicos e Científicos, 1998.

ARTHUR, Lowell Jay. **Melhorando a qualidade do software –** um guia completo para o TQM. Rio de Janeiro: Livraria e Editora Infobook, 1994.

ARTHUR, Lowell Jay. **Produtividade do programador –** um guia para gerentes, analistas e programadores. Rio de Janeiro: Campus, 1985.

BLOOR, Robin. **bloorpaper.doc**. NGSET - The art of software transformation. dez. 1998. Arquivo (155 kbytes). Disco Rígido. Word 97.

DALMOLIN, Denis Alberto. **Ferramenta de apoio a reestruturação de código fonte em linguagem C++ baseado em padrões de legibilidade**. 2000. 74 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

DATE, C. J. **Guia para o padrão SQL.** Rio de Janeiro: Campus, 1989.

FURLAN, José Davi. **Reengenharia da informação –** do mito à realidade. São Paulo: Makron Books, 1994.

NRC - U.S. Nuclear Regulatory Commission Home Page. **Generic safe programming attributes – maintainability,** Maryland, 2000. Disponível em: <http://www.nrc.gov/NRC/NUREGS/CR6463/ch2.htm>. Acesso em: 11 nov. 2000.

ORACLE CORPORATION. **ch1.htm**. Oracle8 SQL reference - release 8.0. 1997a. Arquivo (15,9 kbytes). CD-Rom. Quadralay WebWorks Publisher 3.5.0.

ORACLE CORPORATION. **preface.htm**. PL/SQL user's guide and teference. 1997b. Arquivo (24,9 kbytes). CD-Rom. Quadralay WebWorks Publisher 3.5.0.

PARIKH, Girish. **Reengenharia de software:** técnicas de manutenção de programas e sistemas. Rio de Janeiro: Livros Técnicos e Científicos Ed., 1990.

STAA, Arndt von. **Programação modular:** desenvolvendo programas complexos de forma organizada e segura. Rio de Janeiro: Campus, 2000.

UNIVERSITY OF NOTRE DAME. **SQL, SQL*PLUS and PL/SQL standards,** Notre Dame, 1996. Disponível em : < http://www.nd.edu/~ndora/standard/sql.html>. Acesso em: 01 abr. 2001.

ZSCHORNACK, Fábio. **HoMe-PaGe Fábio Zschornack**, 1999. Disponível em:<http://minerva.ufpel.tche.br/~fabio/baguall/introd.htm>. Acesso em: 16/04/2001.