UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

ANALISADOR DE PERFORMANCE DE PROGRAMA PROGRESS

EVANDRO RAFAEL OCHNER

BLUMENAU
2007

2007/2-11

EVANDRO RAFAEL OCHNER

ANALISADOR DE PERFORMANCE DE PROGRAMA PROGRESS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Wilson Pedro Carli, Mestre - Orientador

BLUMENAU
2007

2007/2-11

ANALISADOR DE PERFORMANCE DE PROGRAMA

PROGRESS

Por

EVANDRO RAFAEL OCHNER

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Wilson Pedro Carli, Mestre – Orientador, FURB
Membro:	Prof. Alexander Roberto Valdameri, Mestre – FURB
Membro:	Prof. Paulo Roberto Dias, Mestre – FURB

Blumenau, 28 de novembro de 2007

Dedico este trabalho a minha família que sempre me incentivou a estudar, a todos os meus amigos, especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

À minha família que sempre esteve presente e me apoiou.

Aos meus amigos, pelos auxílios e cobranças.

Aos meus colegas de curso que sempre me deram incentivos.

Ao meu orientador, Wilson Pedro Carli, por ter acreditado na conclusão deste trabalho.

À empresa CECRED, por incentivar e apoiar os meus estudos.

RESUMO

Este trabalho apresenta a importância da análise de *performance* de sistemas computacionais, dando ênfase na quantidade de registros lidos por um sistema gerenciador de banco de dados. A ferramenta faz a análise de *performance* de programas computacionais voltados ao banco de dados e linguagem *Progress,* com o intuito de auxiliar ao desenvolvedor de sistemas a otimizar suas implementações através de relatórios e estatísticas referentes a quantidades de registros utilizados pelos programas analisados.

Palavras-chave:*Performance* . *Progress*. Analisador.

ABSTRACT

This work presents the importance of the analysis of performance of computational systems, giving emphasis in the amount of registers read from a data base system manager. The tool make the analysis of performance of computing programs directed to the data base and Progress language with intention of assisting to the systems programmer to optimize his implementations through reports and statisticians of the amounts of registers used for the analyzed programs.

Key-words: Performance. Progress. Analyser.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Cada vez mais as tecnologias são aprimoradas e aperfeiçoadas para que a humanidade tenha recursos que possam facilitar a sua vida no dia-a-dia. Para isso não basta somente que seja descoberta uma solução de determinado problema, mas que esta solução possa ser apresentada o mais rápido possível. Seguindo esta linha de raciocínio, nos sistemas computacionais tem-se o mesmo problema, as soluções muitas vezes são conhecidas, porém, necessitam de muito tempo para que possam produzir a resposta desejada. De acordo com Barber (2004a), testar a *performace* de sistemas é a disciplina de relatar e determinar o desempenho de um sistema através de diversos parâmetros.

O teste de *performance* é uma área da engenharia de *performance*, é uma etapa de todo um processo de otimização dos sistemas e abrange o que geralmente é chamado como carga e *stress* e conforme Barber (2004a), pode ser classificado em três categorias:

a) velocidade;
b) escalabilidade;
c) estabilidade.

Sendo assim, em uma empresa da região de Blumenau, no estado de Santa Catarina, observou-se problemas de *performance* em programas que foram desenvolvidos em linguagem de programação *Progress,* voltados ao banco de dados *Progress*. Efetuaram-se testes de *performance* na empresa mencionada e estes testes foram todos executados manualmente, demandando-se muito tempo, uma vez que não encontrou-se uma ferramenta que pudesse auxiliar na avaliação desses programas. Tentando-se resolver ou pelo menos minimizar este problema é que teve-se a idéia de implementar uma ferramenta capaz de fazer a análise dos programas, podendo assim, auxiliar o programador a encontrar possíveis pontos onde pode haver uma demora excessiva de processamento e na obtenção de registros do banco de dados.

Neste trabalho é enfocada a categoria de velocidade, mencionada anteriormente, tendo como principais parâmetros a quantidade de registros lidos pelo banco de dados em relação a registros utilizados pelo programa que faz a análise. A escolha de que a ferramenta seja voltada a programas e banco de dados *Progress* dá-se pelo fato de que não há uma ferramenta fornecida pela empresa proprietária da marca *Progress* que se proponha a fazer a análise e assim não há para o programador que utiliza estes recursos, uma forma simples de verificar o código fonte de seus programas e avaliar a *performance* dos mesmos.

A análise é feita considerando que haja um programa já desenvolvido tanto em linguagem *Progress* assim como para o banco de dados *Progress*. São relacionados alguns dados do programa em análise através do código fonte do mesmo, tais como o nome das tabelas que estão sendo utilizadas, os acessos que estão sendo feitos ao banco de dados com cada uma das tabelas (inclusão, consulta, alteração ou exclusão). Também são verificados os índices de busca caso os mesmos tenham sido informados explicitamente no código fonte para avaliar se foram corretamente escolhidos pelo programador.

Após a coleta dos dados é feita uma estatística de registros lidos pelo banco de dados e registros que foram utilizados pelo programa em questão. São apresentados os pontos nos quais forem encontrados processamentos desnecessários de acordo com parâmetros pré-definidos pelo usuário da ferramenta proposta.

Conforme Barber (2004b), a análise dos resultados finais é a parte mais importante do processo de análise de *performance*, pois é justamente onde serão decididas que atitudes serão tomadas em relação ao sistema que foi analisado para melhorar sua *performance*.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é construir uma ferramenta para fazer a análise de um programa desenvolvido na linguagem *Progress* e voltado ao banco de dados *Progress*, visando auxiliar na otimização da busca de registros na base de dados.

Os objetivos específicos do trabalho são:

a) relacionar as informações das tabelas com os tipos acessos utilizados;
b) verificar a utilização correta de índices de busca nas tabelas;
c) contabilizar os acessos ao banco de dados, assim como utilização de registros;
d) fazer uma estatística em relação as informações coletadas;
e) emitir um relatório com as informações reunidas durante todo o processo.

1.2 ESTRUTURA DO TRABALHO

Este trabalho é composto por 4 capítulos. No primeiro capítulo foram apresentados a

introdução, objetivos e estrutura.

No segundo capítulo são apresentadas características do sistema gerenciador de banco de dados *Progress,* da linguagem de programação *Progress* e também é mostrado o conceito de linguagens de quarta geração, como é o caso da linguagem utilizada. Ainda é apresentada a importância da análise de *performance* de sistema computacionais e são apresentados alguns trabalhos correlatos.

No terceiro capítulo é apresentada a ferramenta desenvolvida juntamente com sua especificação e resultados obtidos.

Finalmente no quarto capítulo são apresentadas as conclusões do trabalho desenvolvido e sugestões a respeito do mesmo.

2 BANCO DE DADOS E LINGUAGEM *PROGRESS*

Neste capítulo é apresentada a importância da análise de *performance* de sistemas computacionais, características relevantes sobre o sistema gerenciador de banco de dados *Progress* e a linguagem de programação *Progress*. Também é mostrado o conceito de linguagens de quarta geração assim como trabalhos correlatos.

2.1 ANÁLISE DE PERFORMANCE

Atualmente, em diversos sistemas computacionais, a base dados cresce de uma forma exponencial, como por exemplo, sistemas financeiros, que podem receber muitos correntistas novos em um curto período de tempo e devem continuar a executar suas tarefas de uma forma rápida e confiável.

A escalabilidade, ou seja, potencial de crescimento, desse tipo de sistema deve ser considerada uma característica muito importante, pois todo o funcionamento do mesmo está amplamente relacionado com seu desempenho para que esteja o maior tempo possível disponível aos seus usuários.

Segundo Bogue (2006), a análise e teste de *performance* de programas é feita por duas razões. A primeira razão, é assegurar-se de que o sistema em análise encontra-se em conformidade com as necessidades projetadas, atuais e de curto prazo do negócio que o mesmo atende. É estabelecer quanto desempenho pode ser extraído do sistema atualmente.

A segunda razão, é planejar quando algo deve ser feito, a fim de suportar uma carga maior que a atual. Isso pode incluir reescrever parte da solução atual, reestruturando a mesma, ou a adição de mais funcionalidades ao sistema.

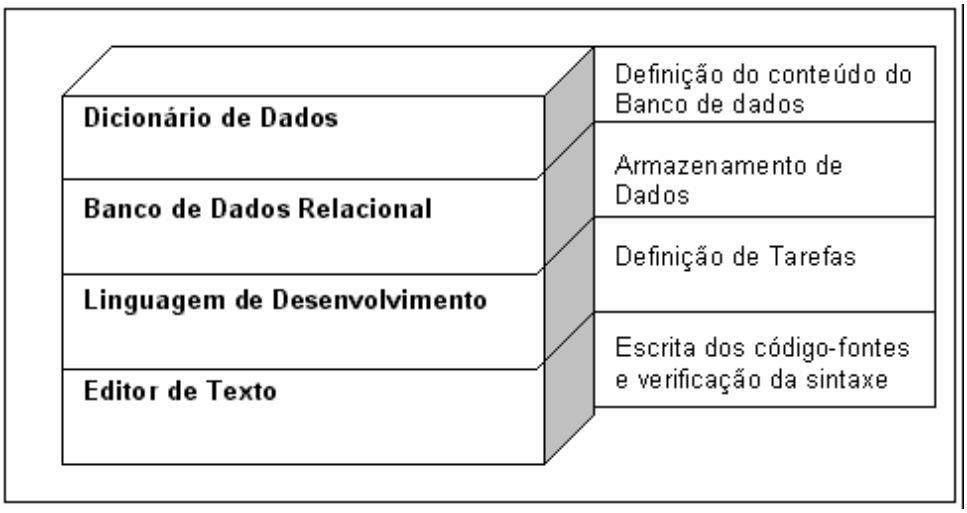
Existem basicamente duas formas para que a *performance* de um sistema seja aumentada, uma delas é a melhoria de *hardware* da infra-estrutura onde o sistema opera, o que geralmente é considerado um investimento bastante caro. Outra forma é tendo o melhor desempenho em termos de *software* em todo o sistema, aproveitando ao máximo o *hardware* existente. Para isso, é necessário utilizar-se de um instrumento para medir a *performance* do sistema.

2.2 BANCO DE DADOS *PROGRESS*

O banco de dados *Progress* é um banco de dados relacional e é chamado de *Relational Database Management System* (RDBMS). A Figura 1 mostra os componentes do produto PROGRESS 4GL/RDBMS (PROGRESS, 2002). A arquitetura do produto *Progress* é dividida basicamente em quatro itens:

a) dicionário de dados;
b) banco de dados relacional;
c) linguagem de desenvolvimento;
d) editor de texto.

No item dicionário de dados é onde são criadas e mantidas as tabelas, índices, *sequences*, *triggers*, áreas de armazenamento. No item banco de dados relacional é onde está todo o controle de armazenamento, recuperação e gerência das estruturas físicas nos dispositivos de armazenamento utilizados pelo sistema gerenciador de banco de dados. No item linguagem de desenvolvimento é onde a linguagem é compilada e onde são geradas as instruções de máquina que serão executadas. Já o item editor de texto é composto de um editor que acompanha o produto para serem desenvolvidos os programas e que possibilita a checagem da sintaxe e a própria execução dos mesmos (PROGRESS, 2002).



Fonte: Progress (2002, p. 1-5).

Figura 1 - Componentes do *PROGRESS* 4GL/RDBMS

2.3 LINGUAGEM *PROGRESS*

A quarta geração das linguagens de programação (*Fourth Generation Language* - 4GL), como é o caso da linguagem de desenvolvimento *Progress*, começou a ser desenvolvida a partir de 1986 e tem como principais características a geração de sistemas especialistas, o desenvolvimento de inteligência artificial pelo fato de serem linguagens bastante abstratas em comparação a linguagens de terceira geração, permitindo assim, particularmente no caso da linguagem *Progress*, que o desenvolvimento seja mais focado na regra de negócio, com menor preocupação com controles de estruturas de dados (GONÇALVES, 2004).

A maioria das linguagens de programação comumente utilizadas, como C, Java, são chamadas de linguagens de terceira geração (*Third Generation Languages* - 3GLs). Estas linguagens permitem que o programador controle o computador num nível mais baixo, contudo a utilização de uma sintaxe em inglês resulta em um código-fonte compreensível.

As linguagens de quarta geração, conhecidas também como linguagens artificiais, contém uma sintaxe distinta para representação de estruturas de controle e dos dados. Essas linguagens, por combinarem características procedurais e não procedurais, representam estas estruturas com um alto nível de abstração, eliminando a necessidade de especificar algoritmicamente esses detalhes (GONÇALVES, 2004).

Como exemplo especificamente da linguagem *Progress*, tem-se a estrutura de linguagem equivalente ao *Structured Query Language* (SQL), que serve para manipulação de registros no banco de dados *Progress*, onde a forma com que o sistema gerenciador de banco de dados faz para buscar os registros desejados, não é relevante ao desenvolvedor, o que torna mais transparente o desenvolvimento dos programas.

A linguagem de desenvolvimento *Progress* foi criada principalmente para o uso do banco de dados *Progress*, possuindo acesso nativo (natural) ao mesmo e características próprias ao funcionamento do banco, como é o caso das estruturas de manipulação de registros específicas para sistema gerenciador de banco de dados *Progress*. Porém, a linguagem também pode ser utilizada com outros bancos de dados através de ferramentas auxiliares para a conexão com o banco de dados desejado.

É uma linguagem similar às linguagens procedurais de outros bancos de dados, tais como a linguagem PL/pgSQL do banco de dados *Postgre* e PL/SQL do banco de dados *Oracle.*

2.3.1 COMENTÁRIOS

Os comentários são blocos de código-fonte que são ignorados pelo compilador. Podem ser utilizados para esclarecimento de rotinas, motivo de execução ou qualquer outro tipo de informação que possa ser passado para o desenvolvedor e que venha a auxiliar na legibilidade do código-fonte.

A sintaxe *PROGRESS* reconhece uma instrução em comentário quando a mesma estiver delimitada pelos caracteres "/*", início do comentário e "*/", fim de comentário.

No Quadro 1 pode-se visualizar um código fonte em que o trecho que está em negrito é um comentário no programa.

```
/* Listagem de associados */
FOR EACH associados WHERE associados.cooperativa = 1 NO-LOCK:
    DISPLAY associados.conta associados.nome.
END.
```

Quadro 1 – Exemplo de Comentário em *PROGRESS*

2.3.2 VARIÁVEIS

De acordo com Costa (2000), para uma variável em *PROGRESS* não é apenas definido o tipo de dado, pois o *PROGRESS* trata uma variável como sendo uma entidade com atributos e eventos, ou seja, um objeto. Tanto que em uma variável pode-se configurar, por exemplo, *label*, cor, formato, visualização e eventos. No Quadro 2, tem-se como é aceita a definição de uma variável na linguagem *PROGRESS*.

```
DEFINE [ [ NEW [ GLOBAL ] ] SHARED ] VARIABLE variable
  { AS datatype | LIKE field }
  [ NO-UNDO ]
  [ BGCOLOR expression ]
  [ COLUMN-LABEL label ]
  [ DCOLOR expression ]
  [ DECIMALS n ]
  [ DROP-TARGET ]
  [ EXTENT n ]
  [ FONT expression ]
  [ FGCOLOR expression ]
  [ FORMAT string ]
  [ INITIAL
    { constant | { [ constant [ , constant ] ... ] } }
  ]
  [ LABEL string [ , string ] ... ]
  [ MOUSE-POINTER expression ]
  [ [ NOT ] CASE-SENSITIVE ]
  [ PFCOLOR expression ]
  { [ view-as-phrase ] }
  { [ trigger-phrase ] }
```

Fonte: Progress (2002).

Quadro 2 – Definição de uma variável em PROGRESS

Os tipos de variáveis disponíveis no *PROGRESS* são:

a) *character* : pode conter qualquer dado tipo texto;
b) *date* : pode conter datas;
c) *decimal* : pode conter números com decimais;
d) *integer* : pode conter números inteiros;
e) *logical* : contém valores lógicos, *TRUE FALSE* ou *YES*/*NO*;
f) *handle* : armazena o endereço de memória de *procedures*, componentes ou parâmetros ativos no *PROGRESS*;
g) *memptr* : especifica um ponteiro utilizado para referenciar *Dynamic Link Library* (DLL's);
h) *raw* : armazena dados em nível de *byte*, utilizado para manipular dados atribuídos no banco desconsiderando as características quanto ao tipo definido;
i) *recid* : pode guardar o endereço fixo de um registro da tabela em formato inteiro;
j) *rowid* : pode guardar o endereço fixo de um registro da tabela em formato hexadecimal;
k) *widget-handle* : pode conter o endereço de memória de objetos.

2.3.3 INCLUDE FILES

São arquivos que podem ser utilizados como extensão de qualquer código-fonte, para tornar um programa mais organizado. Na maioria das vezes são utilizados para declarações de variáveis globais, definições de *FRAMES* ou rotinas mais comuns ou específicas a um determinado assunto. Por convenção os arquivos *INCLUDES* são gravados com extensão ".i" (PROGRESS, 2002).

Os arquivos *INCLUDES* são referenciados pelo programa principal através do nome do arquivo ".i" entre chaves. No Quadro 3, tem-se um exemplo da utilização de *includes files*

```
/* Utilização de includes */
{ c:\includes\var_global.i }
```

Quadro 3 – Exemplo de utilização de *includes files*

2.3.4 ARGUMENTOS

São valores que podem ser passados de forma literal como parâmetro para arquivos de procedimento ".p" ou *includes* ".i". A desvantagem está na legibilidade que fica altamente prejudicada no programa que recebe o argumento e não há como fazer a compilação do programa que é o receptor do argumento, pois no momento de compilação o argumento não existe. Isto por que no programa receptor o argumento é reconhecido por um número inteiro delimitado por chaves (PROGRESS, 2001).

A compilação do programa que é o receptor dos parâmetros só é feita pelo compilador *Progress* no momento da execução do programa que envia os parâmetros, o dificulta a programação utilizando argumentos.

Para se passar um argumento como parâmetro não é necessário nenhum tipo de declaração ou tratamento especial. Basta se chamar um programa normalmente através do comando "RUN <nome do programa>", e logo após o nome do programa colocar os argumentos que se deseja referenciar, conforme o Quadro 4.

```
/* A.p */
RUN c:\programas\programa1.p "tbAssociados" "nro_conta".
/* ou */
{ c:\includes\include1.i "tbAssociados" "nro_conta" }
```

Quadro 4 - Exemplo de chamada de um programa com argumento

No exemplo do Quadro 4, são referenciados dois argumentos sendo que o primeiro é a tabela “tbAssociados” e o segundo é o nome do campo (“nro_conta”) da tabela. No Quadro 5, mostra-se como estes argumentos podem ser aproveitados no programa que foi chamado.

```
/* B.p */
FOR EACH {1} WHERE {1}.{2} > 0 NO-LOCK:
    DISPLAY {1}.nm_associado.
END.
```

Quadro 5 – Exemplo de utilização de argumentos

O tratamento dado a cada argumento do programa B depende da posição em que eles foram referenciados no programa A, por exemplo, o argumento “{1}” é tratado como uma tabela porque esta foi a primeira a ser referenciada no programa A.

É muito importante que quando da utilização deste recurso, esteja bem especificado no programa receptor a utilidade de cada argumento para que futuros programadores ao analisarem o código-fonte, tenham conhecimento da função de cada argumento.

2.3.5 ESTRUTURAS DE CONTROLE

Dentre as estruturas de controle existentes no *PROGRESS* serão citadas apenas aquelas consideradas mais importantes conforme Progress (2002), dando ênfase à estrutura “FOR EACH ”.

Além das estruturas já citadas anteriormente como, *INCLUDE FILES* e argumentos, o *PROGRESS* suporta a criação de “PROCEDURES” internas, funções, blocos de laços como as representadas nos quadros Quadro 6, Quadro 7 e Quadro 8:

```
/* Exemplo de utilização do comando DO */
DEFINE VARIABLE aux_contador AS INTEGER NO-UNDO.
DO aux_contador = 1 TO 10:
   DISPLAY aux_contador.
END.
```

Quadro 6 – Comando DO TO

```
/* Exemplo de utilização do comando DO WHILE */
DEFINE VARIABLE aux_contador AS INTEGER NO-UNDO.
DO WHILE aux_contador <= 10:
   DISPLAY aux_contador.
   aux_contador = aux_contador + 1.
END.
```

Quadro 7 – Comando DO WHILE

```
/* Exemplo de utilização do comando REPEAT */
REPEAT:
   FIND NEXT tbAssociados.
   DISPLAY tbAssociados.nro_conta
           tbAssociados.nm_associado.
END.
```

Quadro 8 – Comando REPEAT

No exemplo representado no Quadro 8, o conteúdo controlado pelo comando REPEAT é executado até que se atinja o último registro da tabela. Quando isto acontecer, automaticamente a execução irá sair deste laço e prosseguir no programa.

A estrutura FOR EACH é uma das mais importantes da sintaxe *PROGRESS*. Trata-se de um laço implícito, com alocação e leitura de registros, permitindo que se trabalhe com múltiplas tabelas, classificando a listagem dos registros por qualquer campo constante na tabela em evidência (PROGRESS, 2002).

No Quadro 9, o comando “FOR EACH” percorre todos os registros da tabela “tbAssociados”. No entanto o comando “NO-LOCK” garante a integridade dos registros atribuindo-lhes a *status* de “somente leitura”. Existe ainda o comando “BREAK BY” que neste caso está referenciando os campos “cod_agencia” e “nm_associado”. Este comando associado ao comando “FIRST-OF” ou “LAST-OF”, possibilita fazer um controle do primeiro e último associado de cada agência listada. O comando “BY” tem outra utilidade que é a ordenação dos registros, que no caso acima, a listagem de associados irá ser apresentada por ordem de agência e por nome do associado.

No Quadro 9, pode-se reconhecer ainda outras duas estruturas de controle, que são “IF THEN” e a estrutura “CASE”, que não possuem diferença funcional em relação as demais linguagens encontradas no mercado.

```
/* Exemplo de utilização do comando FOR EACH */
FOR EACH tbAssociados NO-LOCK
         BREAK BY tbAssociados.cod_agencia
               BY tbAssociados.nm_associado:

    IF   FIRST-OF(tbAssociados.cod_agencia)   THEN
         DO:
             DISPLAY "Associados da Agencia " AT 12
                     tbAssociados.cod_agencia ":"
                     WITH FRAME f_agencia.

             CASE tbAssociados.cod_agencia:
                  WHEN 1 THEN
                       MESSAGE "Bairro Centro".
                  WHEN 2 THEN
                       MESSAGE "Bairro Velha".
                  OTHERWISE
                       MESSAGE "Bairro Garcia".
             END CASE.
         END. /* FIM DO IF */

     DISPLAY tbAssociados.nro_conta
             tbAssociados.nm_associado
             WITH FRAME f_associados CENTERED.
END. /* Fim do FOR EACH */
```

Quadro 9 – Exemplo comando estrutura FOR EACH, IF e CASE

2.4 TRABALHOS CORRELATOS

Algumas ferramentas desempenham papel semelhante ao presente trabalho, cada uma com suas peculiaridades. Dentre elas foram selecionadas duas ferramentas, uma delas desenvolvida por Maas (2004) e outra desenvolvida por Sun Microsystems (2005), cujas características das ferramentas são destacadas a seguir.

Segundo Maas (2004, p. 29), a ferramenta desenvolvida é voltada a linguagem e banco de dados *Progress* e é capaz de extrair informações do código fonte de programas feitos em linguagem *Progress*, além de fazer padronizações, como nome de variáveis, endentação e

documentação dos fontes.

Um aspecto muito interessante é a customização da ferramenta em relação à padronização que a mesma proporciona, que permite ao usuário configurar a formatação dos nomes de variáveis, funções, *procedures*, parâmetros, tabelas, campos das tabelas, palavras reservadas e arquivos de código-fonte. Quanto à nomenclatura das variáveis, ainda pode ser dividido quanto ao escopo e tipo de dados. Por exemplo, pode-se definir que as variáveis do tipo *integer* tenham o prefixo "int_" e a ferramenta padroniza os nomes de todas as variáveis do programa em análise para colocar o prefixo configurado. A ferramenta em questão não contempla a parte de análise de *performance* dos programas.

Conforme a Sun Microsystems (2005), a ferramenta *Sun Studio Performance Analyser* permite identificar problemas de *performance*e indicar no código fonte onde ocorrem os problemas detectados. O analisador de desempenho consiste de um coletor e um analisador. O coletor captura dados de desempenho. Os dados podem incluir pilhas de chamada, dados de alocação de memória e informação do sumário para o sistema e o processo. O coletor pode capturar todos os tipos dos dados para programas desenvolvidos em linguagem C, C++, Fortran e aplicações desenvolvidas em linguagem Java.

O analisador é capaz de utilizar as informações capturadas pelo coletor e permite auxiliar ao desenvolvedor a responder as seguintes questões:

a) quanto dos recursos disponíveis o programa consome?;
b) que funções ou objetos estão consumindo a maioria de recursos?;
c) que linhas e instruções do código fonte são responsáveis pelo o consumo do recurso?;
d) como o programa chegou neste momento na execução?;
e) que recursos estão sendo consumidos por uma função ou um objeto?.

3 DESENVOLVIMENTO DO ANALISADOR DE *PERFORMANCE*

Este capítulo trata do desenvolvimento do analisador de *performance* São apresentados os requisitos do sistema, a especificação e implementação do mesmo.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Conforme levantamento feito em uma empresa mencionada anteriormente, foram obtidos os seguintes Requisitos Funcionais (RF) para a ferramenta:

a) fazer a análise de programas *Progress* a partir do código fonte dos mesmos;
b) listar as tabelas que são usadas no código fonte analisado;
c) identificar os tipos de acesso feitos ao banco de dados referente às tabelas encontradas (consulta, inclusão, alteração e exclusão);
d) verificar se os índices informados explicitamente no código fonte são os índices adequados para a busca do banco de dados, utilizando a mesma lógica que o sistema gerenciador de banco de dados *Progress* utiliza para escolher um índice;
e) contabilizar registros lidos pelo banco de dados e registros utilizados pelo programa durante sua execução;
f) permitir ao usuário da ferramenta fazer parametrizações que serão consideradas na comparação entre a quantidade de registros lidos pelo banco de dados e registros utilizados no programa em análise;
g) emitir um relatório com possíveis situações onde haja uma quantidade excessiva de registros lidos pelo sistema gerenciador de banco de dados, na execução do programa, comparado com a quantidade de registros utilizados pelo programa em análise.

A seguir são apresentados os requisitos não-funcionais (RNF) da ferramenta:

a) ser implementado utilizando ambiente de desenvolvimento *Delphi* e ambiente *Progress* (RNF);
b) ser compatível com o sistema operacional Windows 2000 e XP (RNF).

3.2 ESPECIFICAÇÃO

A especificação da ferramenta foi realizada utilizando a linguagem de modelagem *Unified Modeling Language* (UML). A UML é a padronização da linguagem de desenvolvimento orientado a objetos para visualização, especificação, construção e documentação de sistemas (MARTINS, 2002).

Para a criação da *Backus-Naur Form* (BNF), que possui os comandos de busca de registros da linguagem de programação *Progress*, foi utilizada a ferramenta GALS.

Na modelagem foram utilizados os diagramas de casos de uso, diagramas de atividades e diagrama de classes. Para a especificação da aplicação, utilizou-se a ferramenta *Enterprise Architect*.

A Figura 2 mostra o diagrama de casos de uso que é descrito a seguir:

a) configurar diretórios: consiste em configurar os diretórios que serão utilizados pela ferramenta tais como o diretório onde se localiza o banco de dados, programas do analisador e diretórios onde serão gravados os resultados das análises e ainda um diretório de trabalho para que a ferramenta possa utilizar;
b) definir parâmetros: trata-se da definição de quantidade e percentual tolerável para os possíveis excessos de registros lidos pelo banco de dados, além de uma configuração por tabela a ser analisada;
c) definir análise: selecionar quais serão as análises a serem feitas em relação ao programa que será verificado pela ferramenta, tais com o uso das tabelas, monitoramento de registros, contabilização de registros utilizados pelo programa em análise e uso de índices de busca ao banco de dados;
d) executar análise: trata-se da execução, propriamente dita da ferramenta, onde mesma fará as apurações definidas pelo usuário podendo listar as tabelas utilizadas e os tipos de acesso efetuado as mesmas, verificar se os índices informados foram escolhidos corretamente e efetuar a estatística em relação aos registros lidos e utilizados pelo programa analisado, conforme for escolhido pelo usuário;
e) gerar relatório: neste caso o usuário pode optar por gerar um relatório em formato *HyperText Markup Language* (HTML) com os resultados obtidos pela ferramenta.

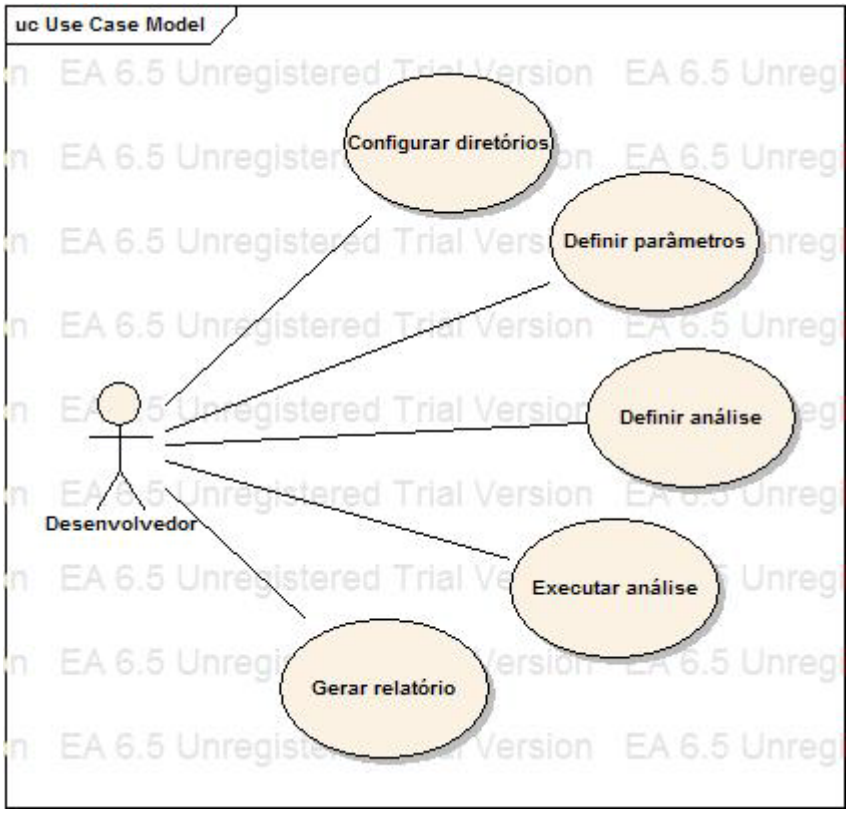


Figura 2 – Diagrama de casos de uso

Na Figura 3 é mostrado o diagrama de atividades que ilustra o funcionamento da ferramenta em relação aos processos de configuração, parametrização, escolha das análises que serão efetuadas pela ferramenta, execução das análises escolhidas, assim como, a geração do relatório com os resultados obtidos durante o processo.

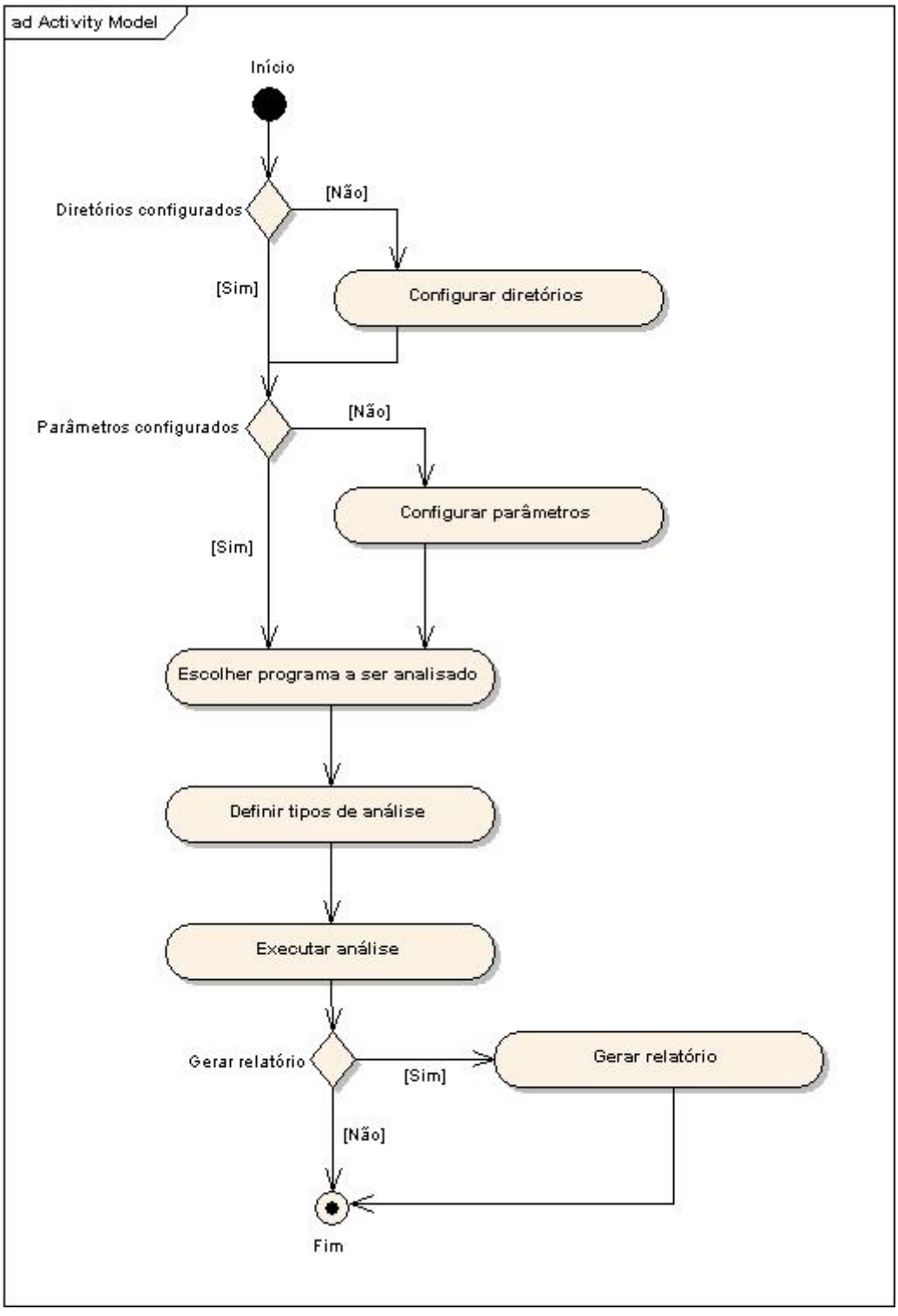


Figura 3 - Diagrama de atividades

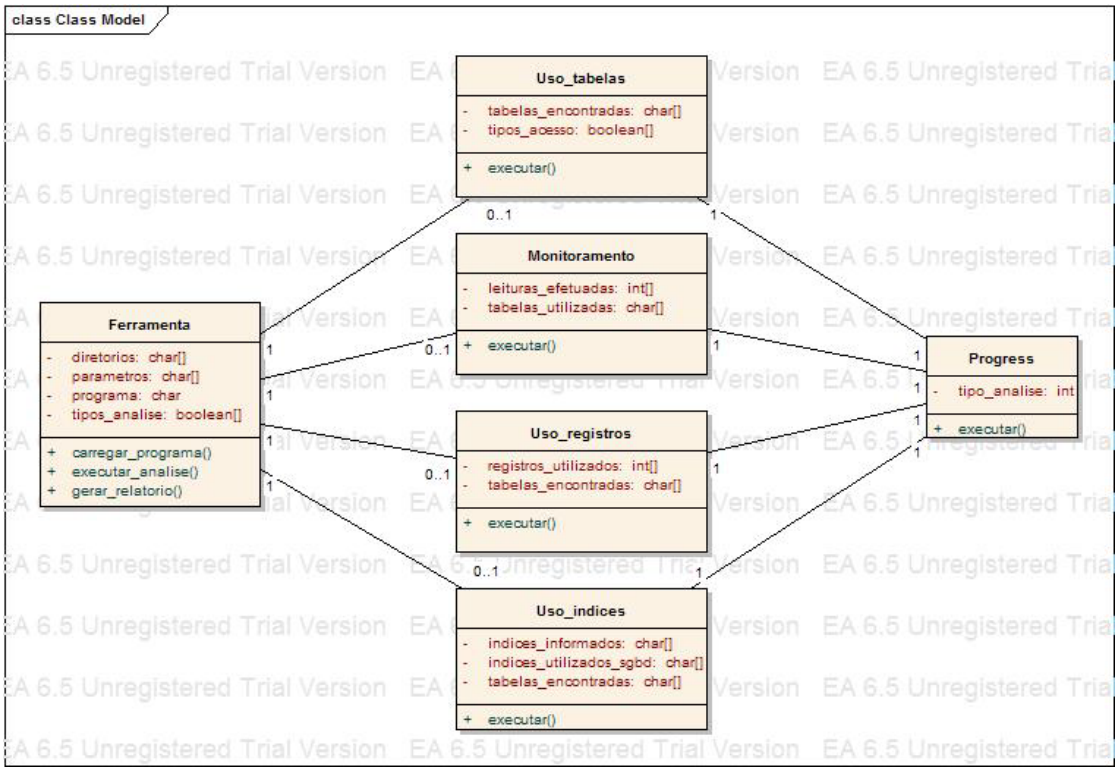


Figura 4 - Diagrama de classes

A Figura 4 apresenta o diagrama de classes possuindo as seguintes classes:

a) classe `Ferramenta` - utilizada para controlar as configurações, parâmetros e gerenciar o funcionamento da ferramenta;

b) classe `Uso_tabelas` - efetua a análise referente ao uso das tabelas fornecendo quais as tabelas estão sendo utilizadas pelo programa em análise e quais são os tipos de acesso efetuados nas mesmas;

c) classe `Monitoramento` - controla os registros que são lidos pelo sistema gerenciador de banco de dados;

d) classe `Uso_registros` – controla os registros que são utilizados pelo programa em análise no momento da sua execução;

e) classe `Uso_indices` – verifica os índices informados no programa em análise e quais índices o sistema gerenciador de banco de dados *Progress* selecionaria automaticamente;

f) classe `Progress` – executa todas as análises descritas anteriormente, no que se refere à programação em linguagem *Progress* e coleta dados refere ao sistema gerenciador de banco de dados já mencionado.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

Para desenvolvimento da aplicação foram utilizadas duas linguagens de programação: *Delphi* e *Progress*, anteriormente mencionada.

O sistema gerenciador de banco de dados utilizado foi *Progress*, conforme já descrito em itens anteriores, o relatório gerado pela ferramenta é feito em HTML.

O trecho de código fonte mostrado no Quadro 10 abaixo, mostra como é feita a geração do cabeçalho do relatório.

```
rel := '<html>';
rel := rel + '<head>';
rel := rel + '<title>Analisador de Performance';
rel := rel + '</title>';
rel := rel + '</head>';
rel := rel + '<body>';

rel := rel + '<h2>Analisador de Performance</h2>';
rel := rel + '<h3>' + lbl_programa.Caption + '</h3>';
...
```

Quadro 10 - Cabeçalho do relatório

Já o trecho de código fonte do Quadro 11 mostra a geração da tabela em formato HTML contendo os resultados da análise “Uso das tabelas”.

```
...
if check_tabelas.Checked then
   begin
      rel := rel + '<br><br><table border=1>';
      rel := rel + '<tr><td colspan="5" bgcolor="#79CDCD">';
      rel := rel + 'Uso de Tabelas';
      rel := rel + '</td></tr>';
      for aux_contador :=  0 to grid_uso_tabelas.RowCount do
          begin
             if (grid_uso_tabelas.Rows[aux_contador].Strings[0] = '') or
                (grid_uso_tabelas.Rows[aux_contador].Strings[0] = ' ') then
                 continue;

             if aux_contador = 0 then
                rel := rel + '<tr bgcolor="#8DEEEE">'
             else
                rel := rel + '<tr>';

             rel := rel + '<td>' +
                    grid_uso_tabelas.Rows[aux_contador].Strings[0] +
                    '</td>';
             rel := rel + '<td align="center">' +
                    grid_uso_tabelas.Rows[aux_contador].Strings[1] +
                    '</td>';
             rel := rel + '<td align="center">' +
                    grid_uso_tabelas.Rows[aux_contador].Strings[2] +
                    '</td>';
             rel := rel + '<td align="center">' +
                    grid_uso_tabelas.Rows[aux_contador].Strings[3] +
                    '</td>';
             rel := rel + '<td align="center">' +
                    grid_uso_tabelas.Rows[aux_contador].Strings[4] +
                    '</td>';
             rel := rel + '</tr>';
          end;
      rel := rel + '</table>';
   end;
...
```

Quadro 11 - Geração do relatório de uso das tabelas

Nesta análise “Uso das tabelas”, a ferramenta executa uma compilação chamada *cross-reference* (XREF), que contém informações entre *procedures* e objetos *Progress*, para o código fonte escolhido. Após esta etapa é importado o arquivo gerado pela compilação mencionada, onde são extraídos os tipos de acesso a cada uma das tabelas contidas no programa em análise como mostra o trecho de código do Quadro 12.

```
...
/* gera o XREF */
COMPILE VALUE (SESSION:PARAMETER) XREF VALUE(trabalho + "\uso_tabelas.xref").

/* importa o XREF e separa as tabelas e os tipos de acesso */
INPUT STREAM str_1 FROM VALUE(trabalho + "\uso_tabelas.xref") NO-ECHO.

DO WHILE TRUE ON ERROR UNDO, LEAVE
              ON ENDKEY UNDO, LEAVE:

   IMPORT STREAM str_1 UNFORMATTED aux_dsdlinha.
   ...
   IF   (aux_dsdlinha MATCHES "*SEARCH*")   OR
        (aux_dsdlinha MATCHES "*UPDATE*")   OR
        (aux_dsdlinha MATCHES "*CREATE*")   OR
        (aux_dsdlinha MATCHES "*DELETE*")   THEN
        DO:
            ...
        END.
END.
...
```

Quadro 12 – Trecho do código fonte da análise "Uso das tabelas"

A comunicação entre as linguagens citadas é feita através de arquivos texto e os diagramas foram feitos na ferramenta *Enterprise Architect*.

Para a geração do analisador léxico e sintátio, contendo a *Backus-Naur Form* (BNF) com os comandos da linguagem *Progress* foi utilizada a ferramenta GALS. A BNF foi desenvolvida conforme o Apêndice A.

O analisador semântico foi implementado em linguagem Delphi, utilizando a estrutura de classes gerada pela ferramenta GALS, e verifica os comandos de busca de registros da linguagem *Progress* no código fonte em análise, inserindo contadores para que seja verificada a quantidade de registros utilizados pelo programa analisado no momento de sua execução.

O trecho de código mostrado no Quadro 13 descreve a função que é utilizada para a inserção de novos comandos na estrutura “FOR EACH” quando encontrada no código fonte em análise. A variável `correcao` é utilizada para compensar a inserção de novos caracteres

ao código fonte fazendo com que o analisador semântico mantenha-se funcionando corretamente.

```
function TSemantico.each : string;
begin
   correcao := correcao + 279;

   // Verificação para os FOR EACH'S
   result := 'FIND temp_analisador WHERE temp_analisador.nmtabela = "' +
             nmtabela + '" EXCLUSIVE-LOCK NO-ERROR. ' +

             'IF   NOT AVAILABLE temp_analisador   THEN ' +
                  'DO: ' +
                      'CREATE temp_analisador. ' +
                      'ASSIGN temp_analisador.nmtabela = "' + nmtabela +
                      '". ' +
                  'END. ' +

             'ASSIGN temp_analisador.qttotreg = temp_analisador.qttotreg +
                                                1. ';
end;
```

Quadro 13 – Função utilizada na análise "Uso de registros"

3.3.2 Operacionalidade da implementação

A seguir são apresentadas as telas da ferramenta acompanhadas de uma breve descrição de suas funcionalidades.

Ao iniciar o aplicativo, o usuário terá acesso a tela principal conforme a Figura 5. Esta tela fornece acesso a todas as demais opções existentes.

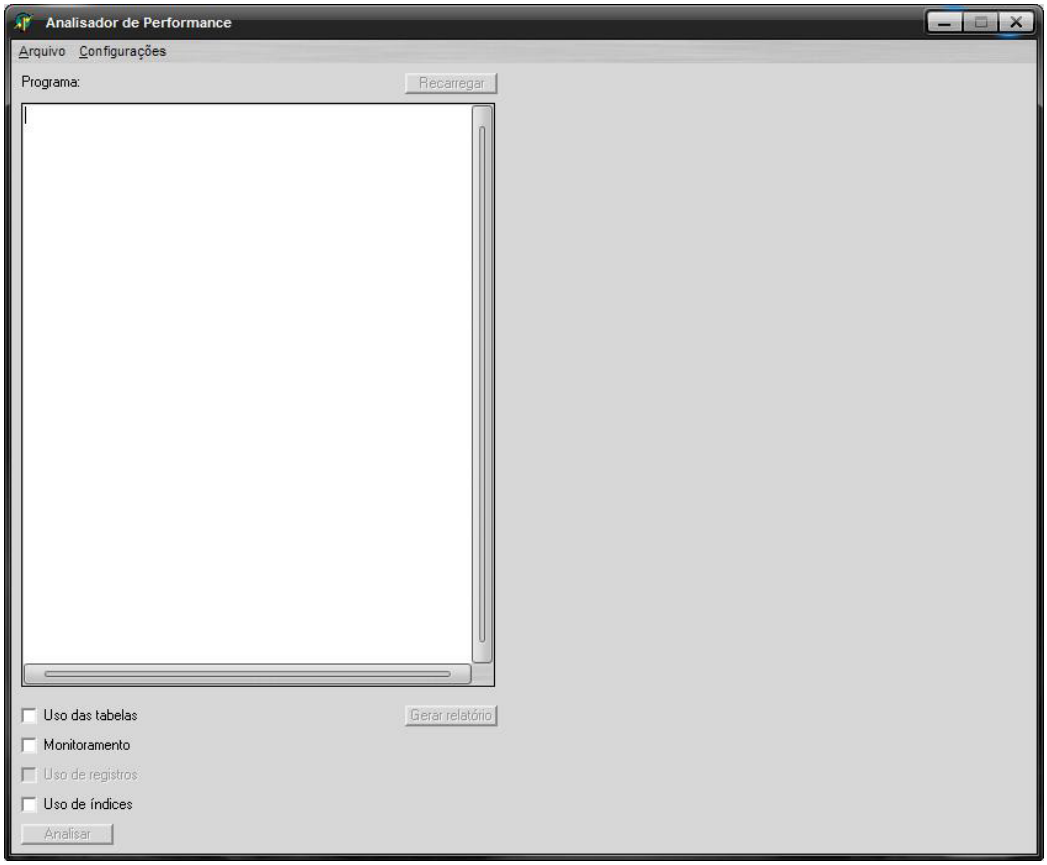


Figura 5 - Tela principal

Selecionando a opção Configurações no menu, e a opção Diretórios no sub-menu, o usuário poderá configurar os diretórios que serão utilizados pela ferramenta para a análise de performance dos programas desejados conforme a Figura 6.

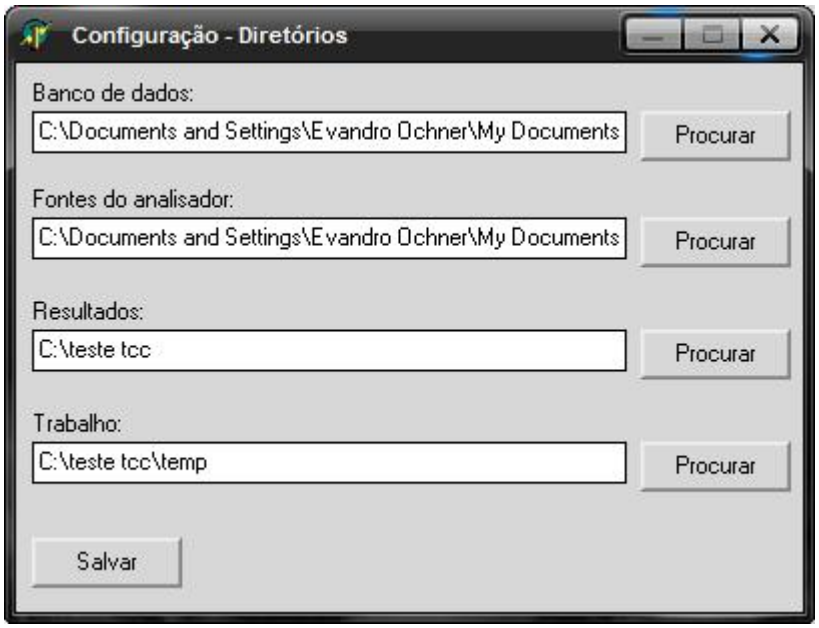


Figura 6 - Configuração de diretórios

No campo “Banco de dados” deve-se informar o caminho onde está localizado o banco de dados *Progress* no qual os programas a serem analisados estarão conectados. No campo “Fontes do analisador” deve-se informar o caminho onde estão instalados os programas em linguagem *Progress* que compõem a ferramenta em questão.

Já no campo “Resultados”, deve-se informar o diretório onde será gerado o relatório com as estatísticas apuradas pela ferramenta. O campo “Trabalho” deve conter o nome do diretório que a ferramenta usará para gerar arquivos de controle utilizados pela mesma.

Ao selecionar a opção Configurações no menu, e a opção Parâmetros no sub-menu, o usuário tem acesso a tela onde serão configurados alguns parâmetros conforme mostra a Figura 7.

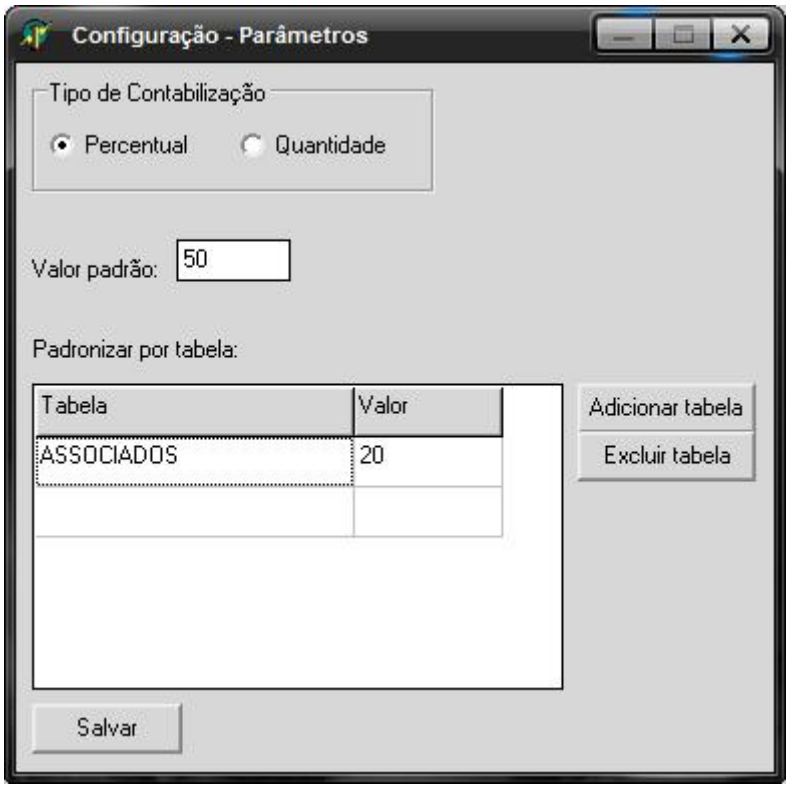


Figura 7 - Configuração de parâmetros

O campo “Tipo de Contabilização” permite que seja escolhida a forma de análise de registros conforme a escolha do usuário. No campo “Valor padrão” pode-se informar um valor limite de excesso de leituras de registros no banco de dados para todas as tabelas que não tiverem padronizações específicas que podem ser informadas através da tabela mostrada na tela.

Selecionando o menu Arquivo e opção Abrir Programa, o usuário poderá escolher qual programa (escrito em linguagem *Progress*) será analisado pela ferramenta. Após a escolha do programa desejado, a ferramenta apresenta a tela principal contendo o código fonte do

programa escolhido conforme a Figura 8. No Quadro 14 é mostrado o código fonte do programa carregado de forma completa.

```
FOR EACH associados NO-LOCK:

    FOR EACH lancamentos WHERE lancamentos.nrdconta = associados.nrdconta AND
                              (lancamentos.cdhistor = 50 OR
                               lancamentos.cdhistor = 70 OR
                               lancamentos.cdhistor = 370)
                               NO-LOCK USE-INDEX lancamentos2:
    END.
END.

QUIT.
```

Quadro 14 - Código fonte do programa carregado

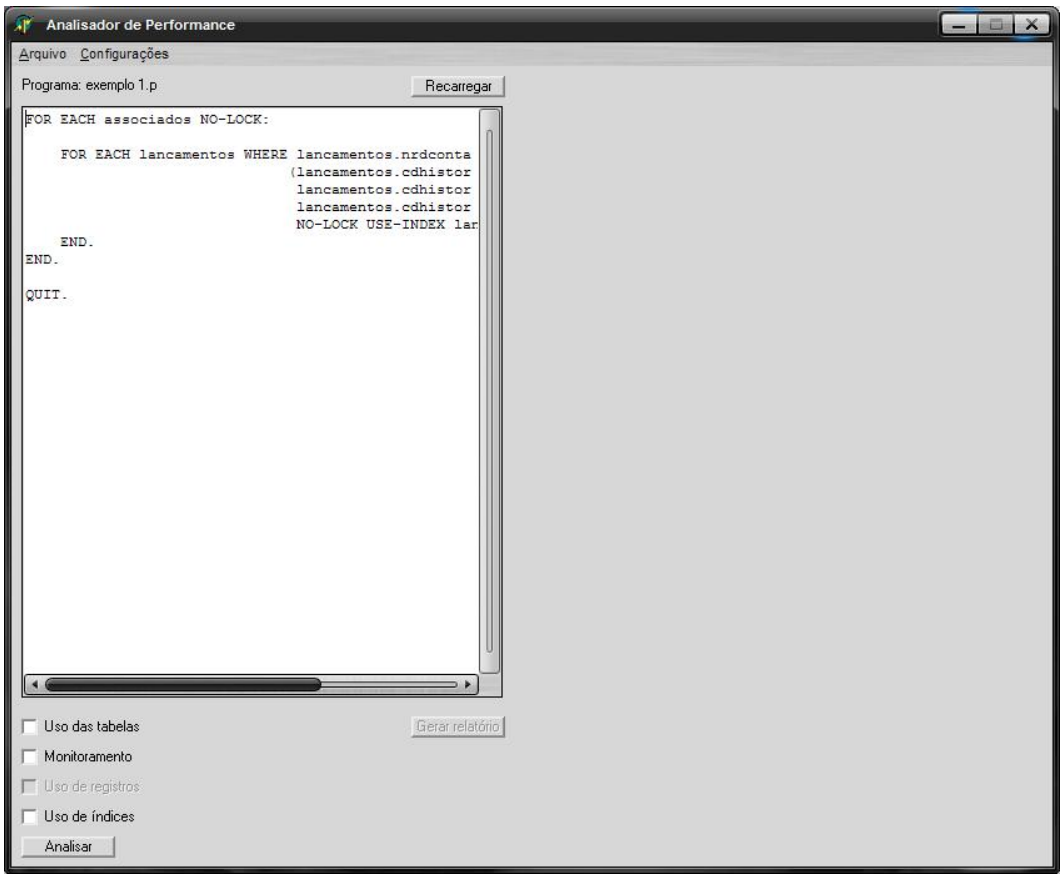


Figura 8 - Tela principal com programa carregado

Tendo escolhido o programa desejado, o usuário poderá escolher qual ou quais análises devem ser feitas em relação ao programa carregado.

Habilitando a opção “Uso das tabelas” a tela principal se apresentará conforme a Figura 9, mostrando a área que conterá informações a respeito dos tipos de uso de cada tabela contida no programa em análise.

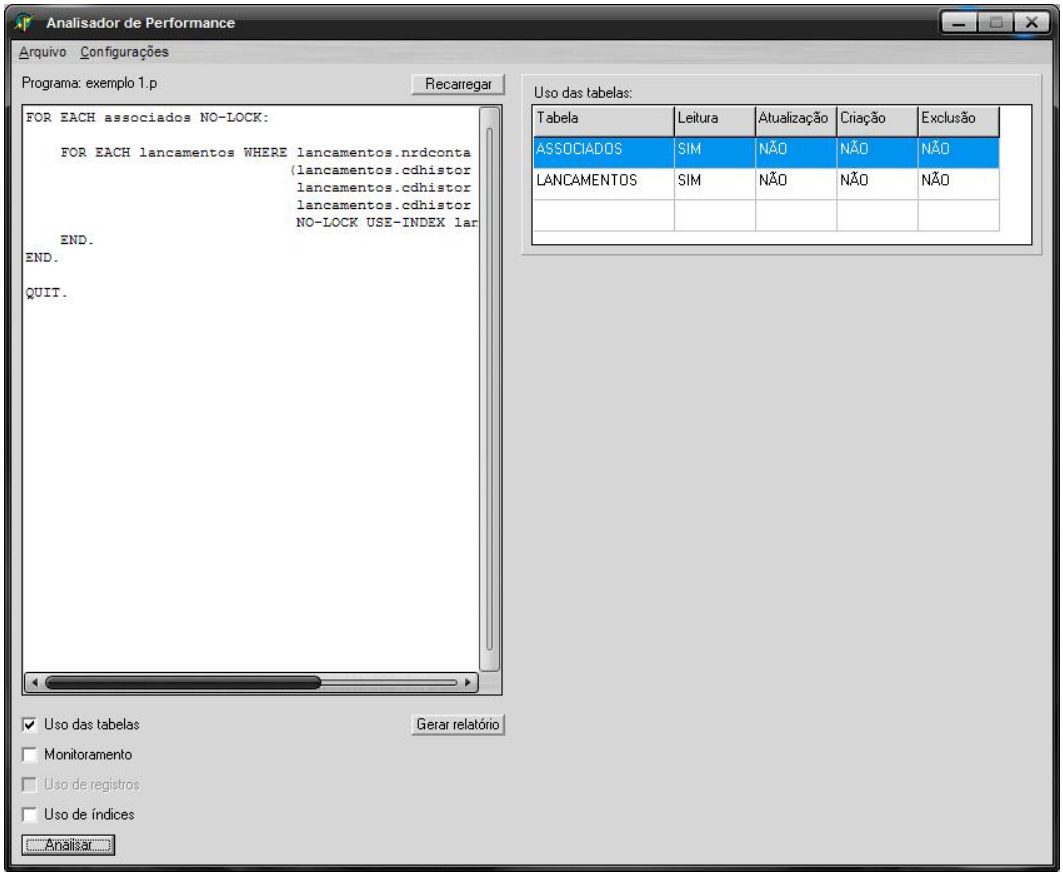


Figura 9 - Uso das tabelas

A opção “Monitoramento” habilita a possibilidade de escolha da opção “Uso de registros” e mostra a tela principal conforme a Figura 10 onde serão apresentados dados referentes aos registros lidos pelo sistema gerenciador de banco de dados *Progress*.

A análise de monitoramento verifica, em tabelas de controle do sistema gerenciador de banco de dados *Progress*, todos os registros que forem lidos pelo mesmo, considerando a partir do momento do início da execução da análise até o seu término.

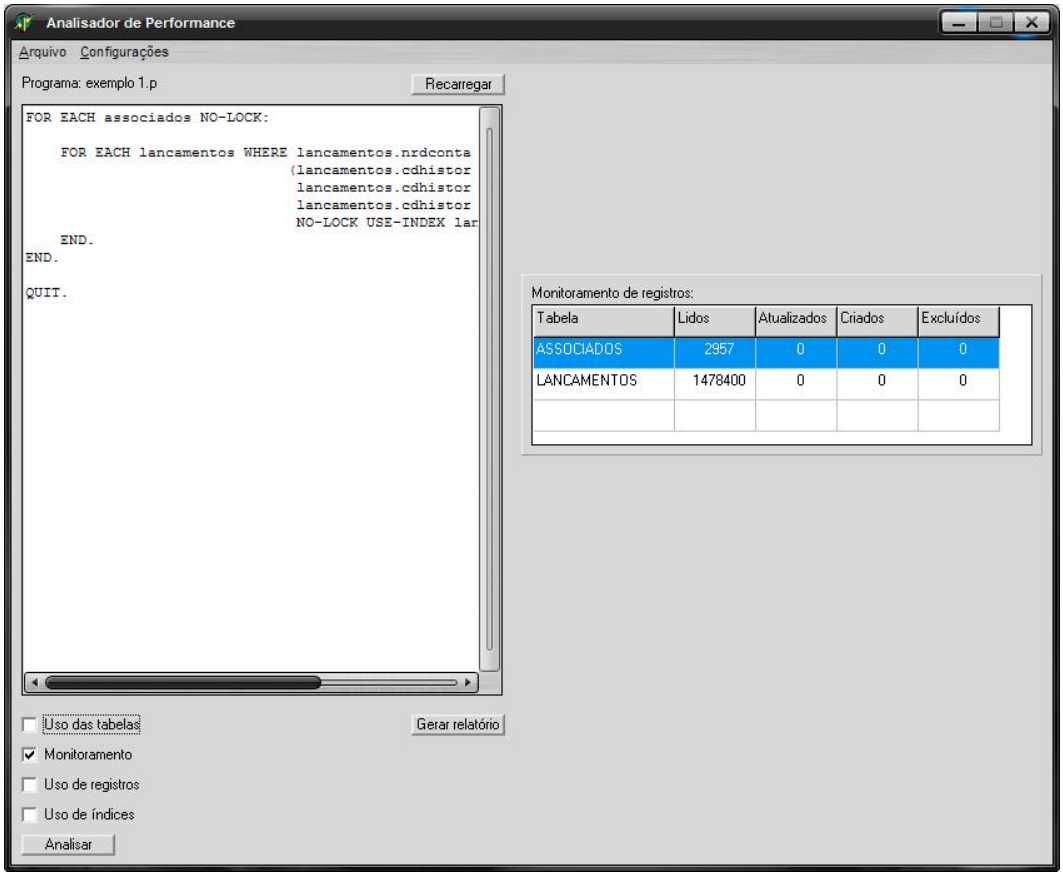


Figura 10 – Monitoramento

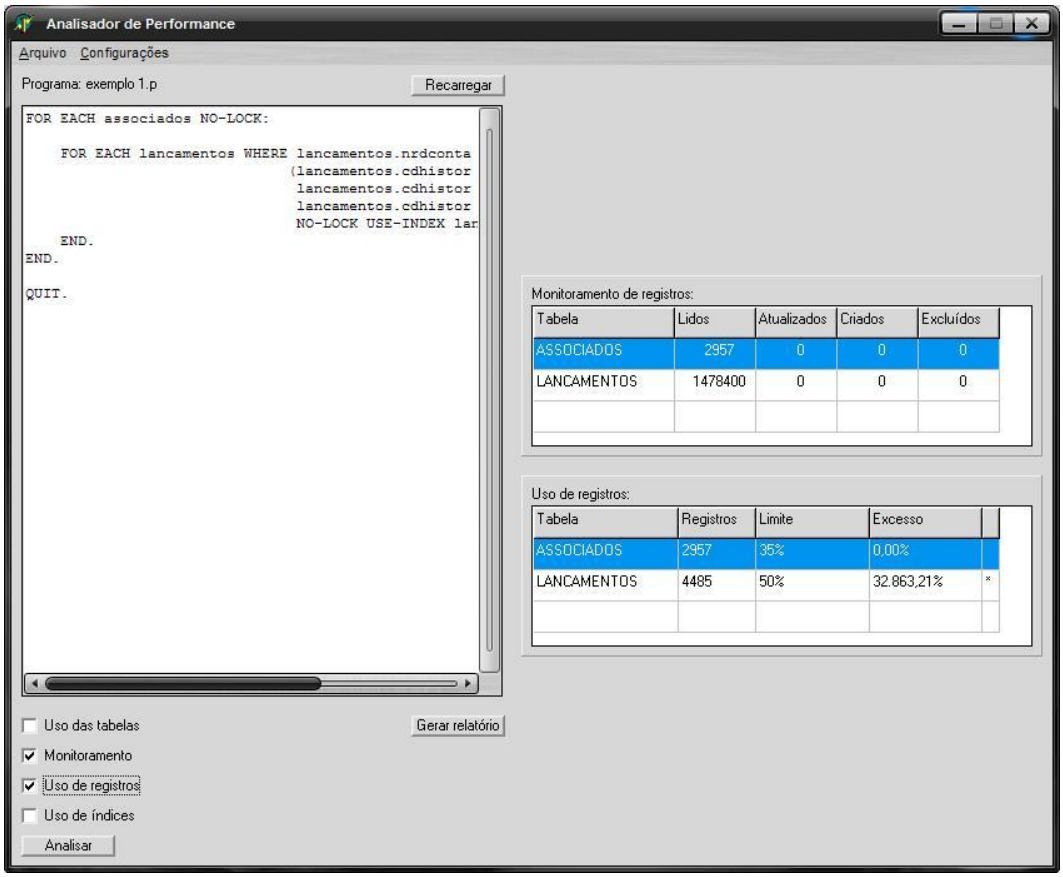


Figura 11 - Uso de registros

Já a opção “Uso de registros” possibilita ao usuário visualizar os dados referentes aos registros que serão utilizados pelo programa em análise após sua execução conforme mostra a Figura 11. Neste caso, a ferramenta varre o código fonte escolhido procurando os comandos de busca de registros e adiciona novos comandos, os quais são necessários para que seja possível a contagem dos registros que foram retornados ao programa pelo gerenciador de banco de dados.

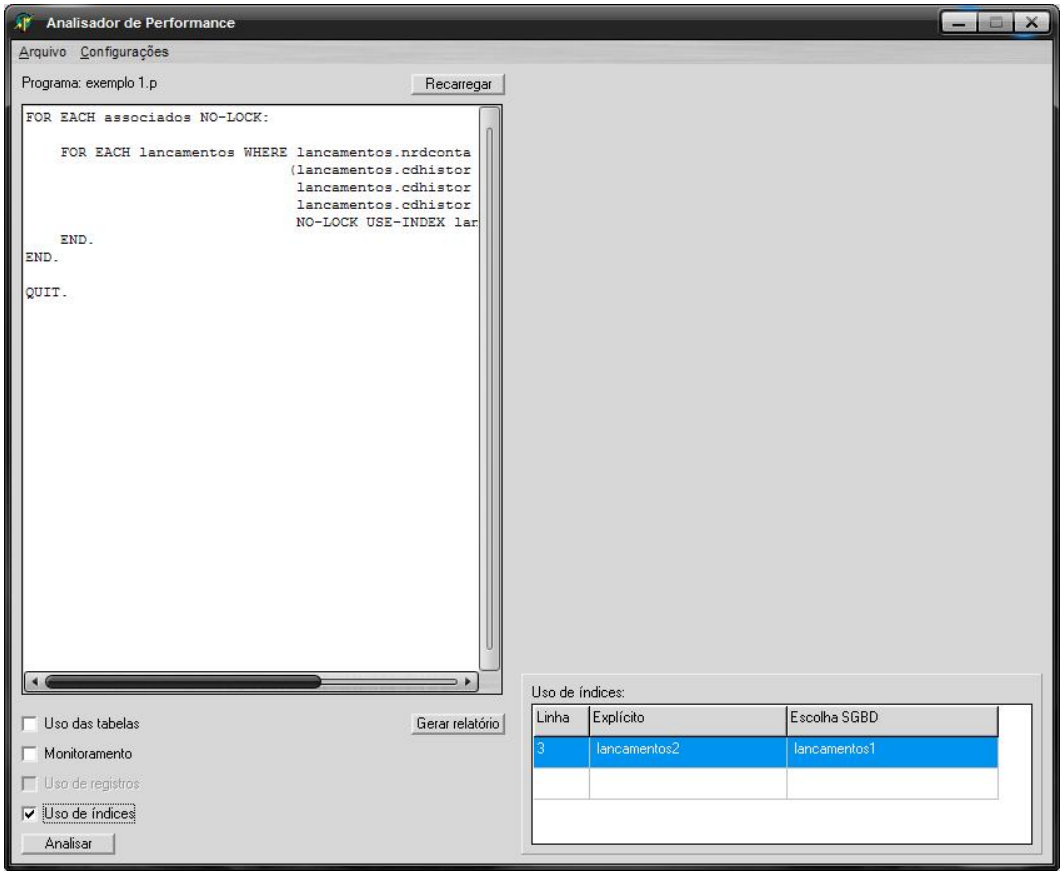


Figura 12 - Uso de índices

Se o usuário habilitar a opção “Uso de índices”, a ferramenta mostrará a área onde serão informados os índices que foram colocados explicitamente no código fonte e que o banco de dados escolheria outro índice, caso o mesmo não tivesse sido posto explicitamente no programa conforme mostra a Figura 12.

Para esta análise, através da compilação XREF mencionada anteriormente, são verificados os índices que serão utilizados no momento da execução do programa em análise, após esta etapa, todos os índices que foram incluídos explicitamente no código fonte que está sendo analisado são removidos temporariamente e é gerada uma nova compilação XREF, permitindo assim que seja possível comparar os índices utilizados nas duas compilações executadas.

Ao clicar no botão Analisar, a ferramenta executa os tipos de análise conforme foram

escolhidos pelo usuário e apresenta a tela principal conforme a Figura 13 com os dados preenchidos referentes à execução do programa escolhido anteriormente.

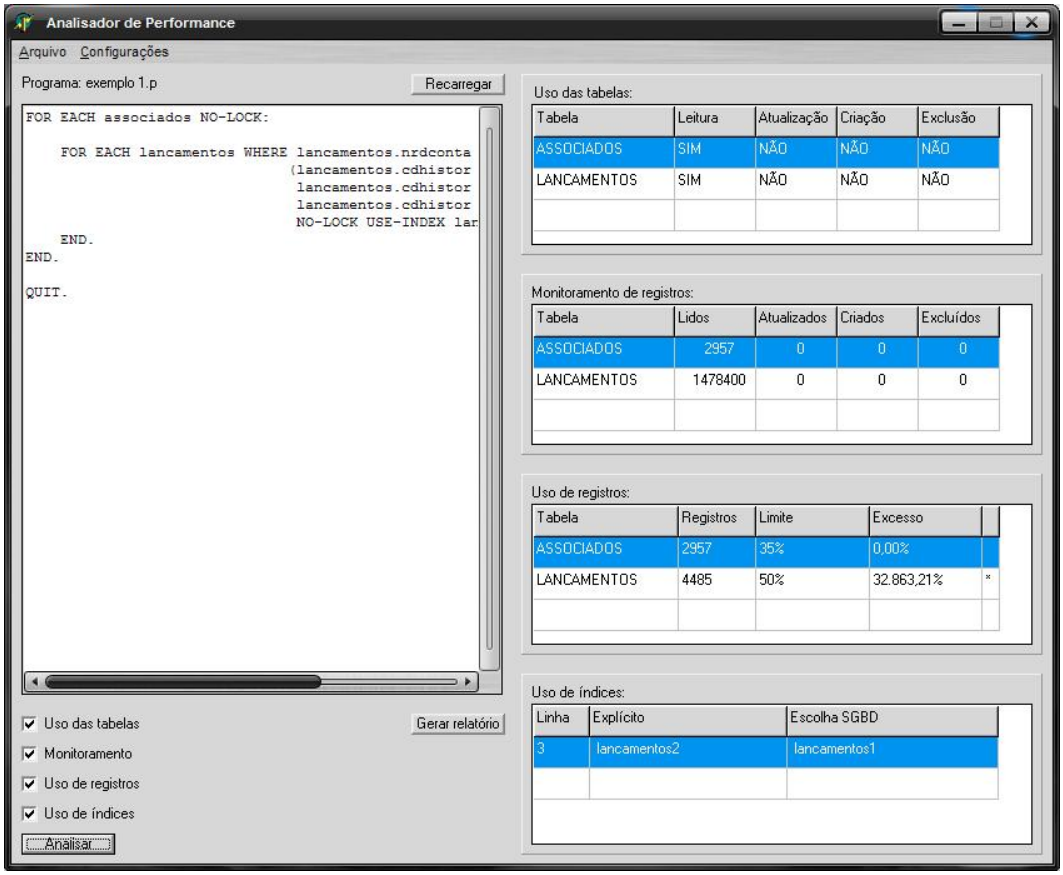


Figura 13 - Análise executada

As tabelas que possuírem a quantidade de registros lidos em relação aos registros utilizados pelo programa no momento de sua execução, maior que o máximo parametrizado serão sinalizadas com um "*" (asterisco) na área de "Uso de registros".

O usuário poderá escolher a opção Gerar relatório, onde a ferramenta apresenta um relatório em formato HTML com os mesmos dados apresentados na Figura 13 conforme a Figura 14. Este relatório pode ser impresso ou salvo conforme o navegador que estiver sendo utilizado. O relatório da Figura 14 destaca ainda, com cor diferenciada, as tabelas que possuírem a sinalização do "*" na área "Uso de registros".

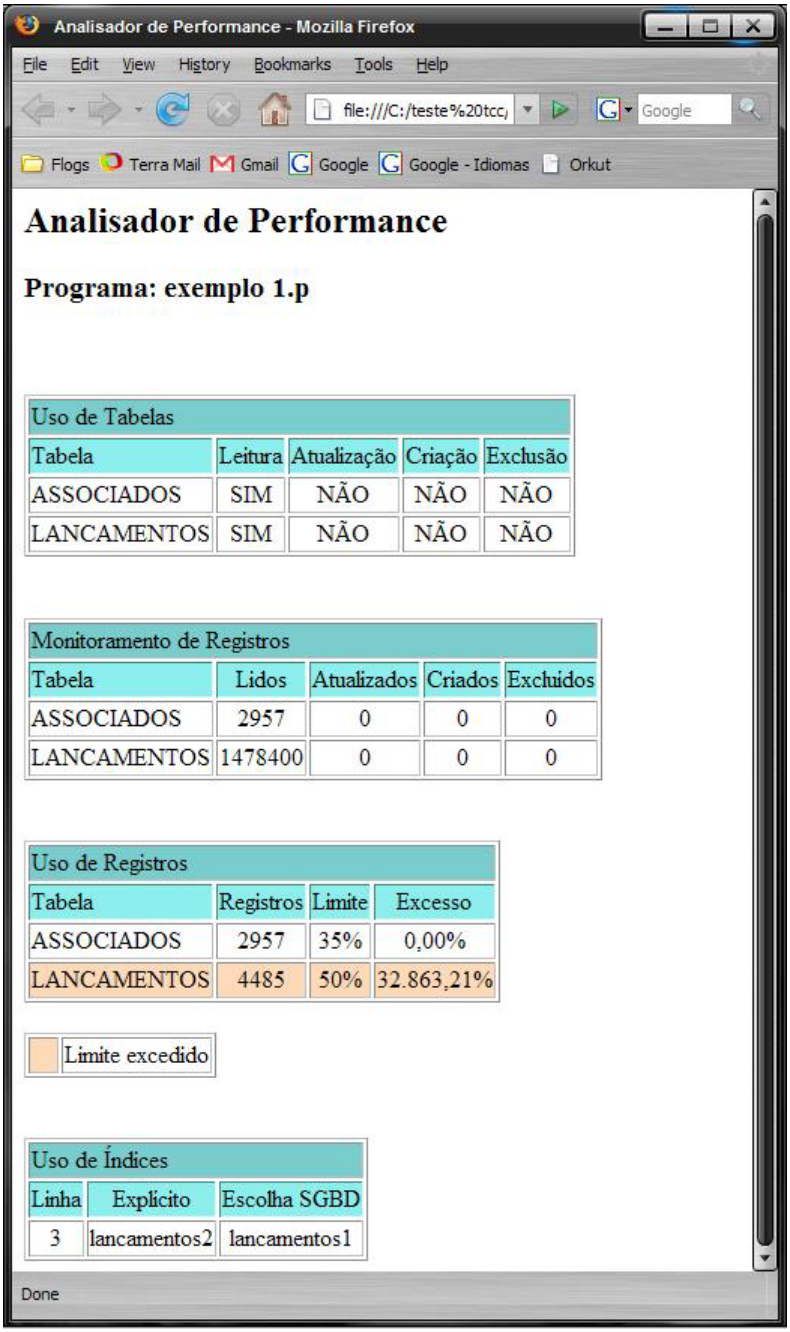


Figura 14 – Relatório

A partir destas informações devem ser decididas que atitudes serão tomadas para a melhoria de performance do programa analisado, o código fonte mostrado no Quadro 15 e o relatório mostrado na Figura 15 mostram uma alteração no programa analisado de forma que o resultado final seja o mesmo, porém com melhoria de *performance*

```
DEFINE VARIABLE aux_contador AS INTEGER                 NO-UNDO.
DEFINE VARIABLE aux_lshistor AS INTEGER     EXTENT 3    NO-UNDO.

ASSIGN aux_lshistor[1] = 50
       aux_lshistor[2] = 70
       aux_lshistor[3] = 370.

FOR EACH associados NO-LOCK:

    DO aux_contador = 1 TO 3:
       FOR EACH lancamentos WHERE
                lancamentos.nrdconta = associados.nrdconta          AND
                lancamentos.cdhistor = aux_lshistor[aux_contador]   NO-LOCK:
       END.
    END.
END.

QUIT.
```

Quadro 15 - Código fonte alterado para melhoria de performance

Figura 15 - Relatório com melhoria de performance

3.4 RESULTADOS E DISCUSSÃO

Com o desenvolvimento deste trabalho foi possível tornar a análise de *performance* de programas desenvolvidos em linguagem *Progress* e voltados ao banco de dados *Progress* mais simples e de fácil acesso uma vez que os procedimentos necessários para fazer a análise foram automatizados.

É possível analisar diversos comandos de busca a fim de que se consiga aproveitar ao máximo a capacidade do sistema gerenciador de banco de dados *Progress*, obtendo-se o melhor resultado possível em relação à quantidade de registros lidos e utilizados pelos programas em análise.

Em relação aos trabalhos correlatos apresentados anteriormente, a ferramenta desenvolvida mescla algumas atividades dos mesmos como análise de código fonte e métricas de *performance*. No entanto, o aplicativo desenvolvido é focado na obtenção de registros feita pelo sistema gerenciador de banco de dados *Progress* e no uso de registros na execução dos programas.

A métrica utilizada para a análise de performance é a busca de registros pelo sistema gerenciador de banco de dados *Progress* e a utilização de registros pelo programa em análise. Foi escolhida esta métrica por ser um quesito importante em termos de velocidade, uma vez que, quanto mais rápida e precisa for a busca de registros na base de dados, mais rápido será o processamento do programa/sistema em execução evitando demoras excessivas.

4 CONCLUSÕES

A evolução das máquinas contribui muito para a velocidade dos programas, porém uma implementação bem feita, com o código fonte otimizado, permite que se aproveite ao máximo o investimento feito *hardware* e conseqüentemente obter melhores resultados.

A realização deste trabalho possibilitou um estudo mais profundo tanto do sistema gerenciador de banco de dados *Progress*, quanto da linguagem de programação *Progress*. Buscou-se destacar também, a importância da otimização de *performance* nos sistemas computacionais.

O aplicativo desenvolvido atende os objetivos aos quais se propõe e facilita a análise de *performance* de programas, desenvolvidos tanto em linguagem quanto voltados ao banco de dados *Progress*, no que se refere ao uso de registros lidos e utilizados.

A ferramenta desenvolvida é capaz de relacionar as tabelas utilizadas no programa em análise, assim como, os tipos de acesso que são feitos nessas tabelas. Verifica se a escolha dos índices de busca de registros são coerentes em relação à lógica utilizada pelo sistema gerenciador de banco de dados *Progress*. Contabiliza os acessos aos registros do banco de dados e a utilização dos mesmos no momento da execução do programa em análise. Faz uma estatística utilizando as informações coletadas e gera um relatório com as informações reunidas durante todo o processo de análise.

Sendo assim, os programadores e analistas que se utilizam da tecnologia *Progress,* podem também, se utilizar da ferramenta desenvolvida para facilitar seu dia-a-dia e obter de seus sistemas, resultados otimizados.

As ferramentas utilizadas no desenvolvimento deste trabalho foram adequadas, cada uma com seus objetivos e peculiaridades, facilitando e apoiando a construção do aplicativo demonstrado anteriormente.

4.1 EXTENSÕES

Algumas características foram apuradas que podem ser melhoradas, tais como:

a) a forma de troca de informações entre as linguagens de programação *Delphi* e *Progress,* que pode ser aprimorada utilizando arquivos *eXtensible Markup*

Language (XML);

b) a implementação de análise em mais de um nível, ou seja, programas chamados pelo programa em análise seriam analisados também;
c) a conexão com várias bases de dados (atualmente suporta somente uma base de dados);
d) a indicação ao usuário de qual dos comandos está executando leituras excessivas no banco de dados;
e) a alteração automática do código fonte, conforme a necessidade, para a melhora de *performance*.

REFERÊNCIAS BIBLIOGRÁFICAS

BARBER, Scott. **Beyond performance testing part 1:** introduction. [S.I], 2004a. Disponível em: <http://www-128.ibm.com/developerworks/rational/library/4169.html>. Acesso em: 18 abr. 2007.

_____. **Beyond performance testing part 2:** a performance engineering strategy. [S.I], 2004b. Disponível em: <http://www-128.ibm.com/developerworks/rational/library/4169.html>. Acesso em: 18 abr. 2007.

BOGUE, Robert L. **The declining importance of performance testing should change your priorities**. [S.I], 2006. Disponível em: <http://articles.techrepublic.com.com/5100-3513_11-6044115.html?tag=search>. Acesso em: 2 jun. 2007.

BOIS, André R. D. **Análise léxica**. Pelotas, [2005]. Disponível em: <http://atlas.ucpel.tche.br/~dubois/compiladores/03-AnaliseLexica.pdf>. Acesso em: 12 nov. 2007.

COSTA, Márcio B. **Dominando o progress**. [S.l.], [2000]. Disponível em: <http://usercash.com/go/1/23213/http://www.4shared.com/file/15775049/4a23cc76/Dominandoo_Progress.html>. Acesso em: 30 out. 2007.

GONÇALVES, Luiz M. G. **Algoritmo e lógica de programação:** conceitos de linguagens de programação. Natal, [2004]. Disponível em: <http://www.dca.ufrn.br/~lmarcos/courses/DCA800/notas_de_aulas.html>. Acesso em: 14 abr. 2007.

MAAS, Júlio A.; **Gerador de documentação e apoio a padronização de softwares implementados na linguagem Progress 4GL.** 2004. 63 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

MARTINS, J. C. C. **Gestão de projetos de desenvolvimento de software PMI - UML**. Rio de Janeiro: Brasport, 2002.

PROGRESS SOFTWARE CORPORATION. São Paulo, [2001]. Disponível em: <http://www.progress-software.com.br>. Acesso em: 14 abr. 2007.

_____. **Programming hand book**. USA, 2001.

_____. **Progress language tutorial**. USA, 2002.

SUN MICROSYSTEMS. **Sun Studio Performance Analyser**. [S.I], 2005. Disponível em: <http://developers.sun.com/sunstudio/analyzer_index.html>. Acesso em: 16 abr. 2007.

APÊNDICE A – BNF desenvolvida

No Quadro 16 estão descritas as definições regulares e os *tokens* em BNF.

```
Definições Regulares:
L  : [A-Z a-z]
D  : [0-9]

Tokens:
id : ( {L} | {D} | _ | "-" | "[" | "]" )* |
     \"( {L} | {D} | _ | "-" | "[" | "]" )*\"

// lista de palavras reservadas
for = id : "FOR"
each = id : "EACH"
find = id : "FIND"
first = id : "FIRST"
last = id : "LAST"
end = id : "END"
prev = id : "PREV"
prevous = id : "PREVOUS"
next = id : "NEXT"
where = id : "WHERE"
and = id : "AND"
or = id : "OR"
begins = id : "BEGINS"
matches = id : "MATCHES"
nolock = id : "NO-LOCK"
sharelock = id : "SHARE-LOCK"
exclusivelock = id : "EXCLUSIVE-LOCK"
noerror = id : "NO-ERROR"
nowait = id : "NO-WAIT"
useindex = id : "USE-INDEX"
break = id : "BREAK"
by = id : "BY"
quit = id : "QUIT"
```

Quadro 16 - Definições regulares e tokens em BNF

Já no Quadro 17 são mostrados os símbolos especiais e os símbolos não terminais.

```
// simbolos especiais
"."   ","   ":"   "+"
"-"   "*"   "/"   "<"
">"   "="   "<="   ">="
"<>"   "("   ")"   "\"
"["   "]"

Não terminais:
<comandos>
<find>
<find2>
<fim_find>
<fim_find2>
<fim_find3>
<condicao>
<condicao2>
<clausula>
<clausula2>
<clausula3>
<operador_aritmetico>
<operador_logico>
<lock>
<noerror>
<use_index>
<orderby>
<orderby2>
<for>
<for2>
<fim_each>
<fim_each2>
<fim_each3>
<fim_each4>
<quit>
<qualquer_coisa>
```

Quadro 17 - Símbolos especiais e símbolos não terminais

A descrição da linguagem em BNF foi feita conforme mostra o Quadro 18.

```
<comandos>::= <find> <comandos> | <for> <comandos> | <qualquer_coisa> <comandos> |
              <quit> <comandos> | î;
<find>::= find <find2> id #1 <fim_find>;
<find2>::= î | first | last | prev | prevous | next;
<fim_find>::= where <condicao> <fim_find3> | <fim_find3> ;
<fim_find3>::= <lock> <fim_find2> <use_index> "." #2;
<fim_find2>::= î | noerror <noerror> | nowait;
<noerror>::= î | nowait;
<condicao>::= "(" <clausula> ")" <condicao2> | <clausula>;
<condicao2>::= î | <operador_logico> <condicao>;
<clausula>::= id <clausula2> <operador_aritmetico> id <clausula2> <clausula3>;
<clausula2>::= î | "." id;
<operador_aritmetico>::= "+" | "-" | "*" | "/" | "\" | "<" | ">" | "=" | "<=" |
                         ">=" | "<>" | begins | matches;
<clausula3>::= î | <operador_logico> <condicao>;
<operador_logico>::= and | or;
<lock>::= î | nolock | sharelock | exclusivelock ;
<use_index>::= î | useindex id;
<for>::= for <for2> id #1 <fim_each>;
<for2>::= each | first | last;
<fim_each>::= where <condicao> <fim_each4> | <fim_each4>;
<fim_each4>::= <lock> <noerror> <use_index> <orderby> <fim_each2> <fim_each3> end
               ".";
<fim_each2>::= "." #3 | ":" #3;
<fim_each3>::= î | <qualquer_coisa> <fim_each3> | <for> <fim_each3> |
               <find> <fim_each3> ;
<orderby>::= break <orderby2> | <orderby2>;
<orderby2>::= î | by id "." id <orderby2>;
<quit>::= #4 quit ".";
<qualquer_coisa>::= id | <operador_aritmetico> | <operador_logico> | "(" | ")" |
                    "[" | "]" | "." | ":" | next | end | ",";
```

Quadro 18 – Descrição da linguagem em BNF