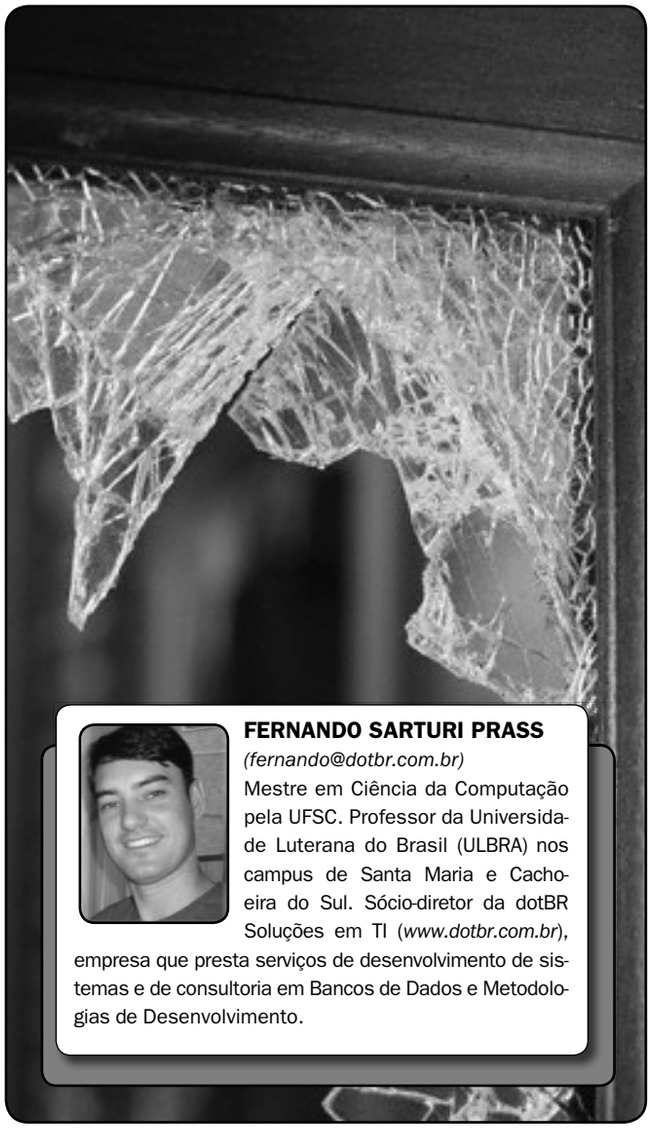
Copiando registros de um banco de dados corrompido



FERNANDO SARTURI PRASS
(fernando@dotbr.com.br)
Mestre em Ciência da Computação pela UFSC. Professor da Universidade Luterana do Brasil (ULBRA) nos campus de Santa Maria e Cachoeira do Sul. Sócio-diretor da dotBR Soluções em TI (*www.dotbr.com.br*), empresa que presta serviços de desenvolvimento de sistemas e de consultoria em Bancos de Dados e Metodologias de Desenvolvimento.

Recentemente a empresa na qual sou consultor foi contratada pela prefeitura de uma grande capital para prestar treinamento e consultoria em Firebird. O principal trabalho que deveria ser realizado durante a consultoria era o de otimização do banco de dados da instituição.

Para ter uma idéia do tamanho do desafio, o banco tinha aproximadamente 120 milhões de registros armazenados em pouco mais de 30 tabelas, o que gera um arquivo com 19 GB. Como se o volume de informação já não fosse um desafio suficientemente grande, a análise do banco de dados mostrou que o mesmo estava corrompido.

Este artigo descreve o trabalho que foi realizado para que fosse possível copiar as informações do banco de dados corrompido. Antes disso, mostra rapidamente qual o ambiente encontrado e os principais passos que foram tomados antes da cópia dos dados.

Avaliação da estrutura

O primeiro passo a ser feito sempre que se deseja melhorar o desempenho de uma aplicação é avaliar o servidor (*hardware*) em que ela está rodando. Muitas vezes comprar um novo equipamento, ou melhorar o existente, não gera o ganho de desempenho esperado, fazendo com que o gasto não se justifique.

O servidor usado pelo cliente tinha dois processadores, 2 GB de memória RAM, discos rígidos de 10 mil RPMs e sistema operacional Windows 2000 Server. Ou seja, o equipamento era de alta qualidade e havia pouco o que melhorar.

Avaliação das configurações do banco de dados

A versão instalada do Firebird era a 1.5, rodando como *SuperServer*. Embora muitos ainda considerem a versão *Classic* a ideal, testes realizados demonstraram que a arquitetura *SuperServer* é bastante estável em servidores de alta qualidade. Esse

Clube83.indb 12 04.04.07 17:38:00

caso é mais um exemplo.

Avaliamos também a forma de gravação dos dados. O banco de dados estava configurado para usar *cache* de gravação do tipo *write throug* (padrão de instalação do Firebird em plataforma Windows), onde cada operação de escrita é imediatamente passada para que o SO a processe. Em outras palavras, a opção *Enable Forced Writes* estava habilitada, fazendo com que o banco gravasse em disco instantaneamente toda atualização feita nos dados, tornando assim as operações de escrita mais lentas.

Outro ponto avaliado foi o tamanho do *cache* das páginas de dados e de índices. Constatou-se que o tamanho era de 120 MB, dada a quantidade de memória disponível e também o número de usuários conectados, avaliamos que o tamanho estava apropriado.

O penúltimo ponto analisado foi o tamanho das páginas de dados e de índices. Normalmente indicamos aos clientes páginas de dados com 8 Kb. Entretanto, devido ao tamanho do arquivo FDB, optamos por usar páginas de 16 Kb. A lógica é simples, com páginas maiores teremos menos fragmentos de registros divididos, uma melhor utilização dos índices e operações de I/O mais contínuas. Como a mudança no tamanho exige um *backup* e um *restore*, optou-se por fazer essa mudança no final.

Por último foram verificadas as estatísticas das tabelas e índices. Foi verificado que as tabelas mais importantes, aquelas que tinham mais de 20 milhões de registros, estavam fragmentadas, ou seja, não estavam gravadas de maneira sequencial no disco, fazendo com que sejam necessárias várias leituras no disco para obter a informação, causando lentidão.

Uma vez constatada que a mudança no tamanho das páginas de dados e índices e a desfragmentação poderiam melhorar significativamente o desempenho da aplicação, passou-se para a realização de um *backup* para um posterior *restore*. Aí começaram os problemas, pois o banco estava corrompido.

Banco de dados corrompido

Para um banco de dados de 19 GB e com mais de 120 milhões de registros, a situação encontrada foi considerada bastante tranquila:

- Dois registros corrompidos, o que significa dizer que dois registros foram perdidos;
- Duas páginas de dados corrompidas: significa dizer que vários registros foram perdidos, já que uma página de dados armazena *n* registros (o valor de *n* varia de tabela para tabela);
- 161 páginas de índices corrompidas: significa que, embora os dados estejam gravados corretamente no banco de dados, uma consulta pode não encontrar as informações caso a parte corrompida do índice esteja apontando justamente para elas (se o índice for usado na consulta, claro).

As análises realizadas revelaram que a corrupção se deu em apenas uma tabela, que armazenava cerca de 21 milhões de registros. Levando em conta a ocupação média dessa tabela (80%) e o número de registros que podem ser gravados numa página de dados dela (101), estimaram-se que poderiam ter sido perdidos aproximadamente 163 registros, o que poderia causar a não emissão de determinados relatórios, ou ainda pior, a emissão de relatórios com dados incompletos (errados).

Foram feitas diversas tentativas com *softwares* de terceiros para corrigir o banco de dados. Apesar da qualidade dos *softwares* utilizados, após horas de processamento infelizmente todos falharam. O que fazer nessa situação, uma vez que todos os recursos de correção haviam sido tentados sem sucesso?

Resolvendo o problema

A solução encontrada para resolver o problema foi criar um novo banco de dados e migrar os dados (registro a registro) do banco atual para o novo banco, com estrutura idêntica, porém intacto.

A idéia é relativamente simples, desenvolver um programa que, primeiramente, extraia as informações do banco e grave as mesmas em arquivos textos. Posteriormente, após a cópia de todos os registros, transferir os dados armazenados nos arquivos para a nova base de dados.

Após a conclusão do processo de cópia, comparar os dois bancos de dados a fim de verificar se existe correspondência entre o número de registros de cada uma das tabelas (o fluxo do sistema pode ser visto na **Figura 1**). Tudo muito fácil, se não fosse o enorme volume de informações.

Estimou-se, através de testes de I/O no servidor, que o tempo necessário para a realização de todo o processo era de aproximadamente 70 horas. Devido ao tempo demasiadamente longo, o uso de arquivos textos intermediários foi necessário para evitar que o processo tivesse que ser recomeçado do zero caso alguma falha, como falta de energia, ocorresse durante o processo de extração ou de cópia dos dados.

Construindo o sistema

Antes de construir o sistema, uma convenção usada aqui no artigo: sempre que um componente que foi adicionado ao pro-

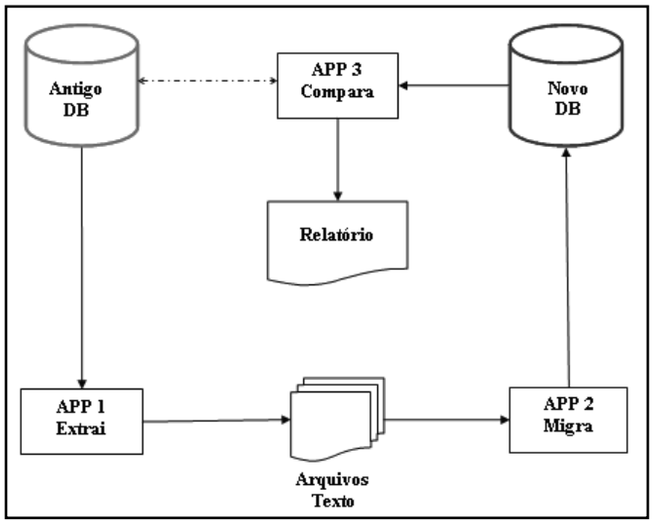


Figura 1. Fluxo geral do processo de migração

Clube83.indb 13 04.04.07 17:38:02

jeto tiver seu nome alterado, o novo nome estará entre parênteses logo após o nome da classe. Por exemplo, onde está escrito "adicionar um *Edit* ("edt_teste")" ler como "adicionar um *Edit* e renomear para "edt_teste"".

Inicie um novo projeto, renomeie o formulário para "frm_principal" e adicione um *Edit* ("edt_pasta_arquivos") que será usado para informar o caminho em que serão armazenados os arquivos texto gerados.

Acrescente também dois componentes *ListBox* ("lb_tabelas_origem" e "lb_tabelas_destino"), um para mostrar todas as tabelas e o outro para as tabelas selecionadas pelo usuário. Adicione quatro *SpeedButtons* ("btn_add", "btn_del", "btn_add_todos" e "btn_del_todos") para a inclusão/exclusão das tabelas e dois *BitBtn* ("btn_migrar_extrair" e "btn_fechar"). O código dos botões (exceto do *btn_migrar_extrair*) está na **Listagem 1**. A interface da tela pode ser vista na **Figura 2**.

Adicione um Data Module ("DM") ao projeto e acrescente ao mesmo dois componentes *IBDatabase* ("db_origem" e "db_destino") e dois *IBTransactions* ("trnsc_origem" e "trnsc_destino"). Faça as devidas configurações e aponte os componentes para os seus respectivos bancos de dados. Adicione um *IBQuery* ("qry") apontando para o *db_origem* e um *IBScript* ("script") apontando para *db_destino*.

Observação: *O autor optou por usar IBX, mas se o leitor preferir pode usar outras bibliotecas, como dbExpress ou IBO.*

Extração/migração dos dados

Acrescente mais um formulário ("frm_execucao") ao programa. Esse formulário fará a extração dos dados, migração, comparação e ainda mostrará o *log* das ações do projeto. Adicione um *Memo* ("log") e dois *BitBtns* ("btn_executar" e "btn_fechar") para executar a operação e fechar o formulário após finalizar o processo (**Figura 3**).

O algoritmo da extração é, resumidamente, o seguinte: para cada uma das tabelas selecionadas pelo usuário o sistema deve obter o nome dos seus campos, selecionar todos os registros da mesma e depois montar uma instrução de INSERT para cada um dos seus registros.

A seguir deve armazenar os INSERTs no(s) arquivo(s) texto. A extração dos dados está em *ExtraiDados* do formulário, cujo código pode ser visto na **Listagem 2**.

Sobre o código: cada arquivo terá no máximo 10.000 registros, toda vez que algo importante acontece, como a criação de um novo arquivo, o *log* é atualizado, isso é feito através do *AtualizaLog* (implementando mais adiante).

Ao ler os dados e montar o INSERT é chamada a função *TransformaDado*, que como o próprio nome sugere, transforma a informação armazenada no campo. Essa transformação

Listagem 1. Código para os botões de inclusão/exclusão de tabelas

```
procedure Tfrm_tabelas.btn_addClick(Sender: TObject);
begin
  if lb_tabelas_origem.ItemIndex > -1 then
    if lb_tabelas_destino.Items.IndexOf(
      lb_tabelas_origem.Items[
      lb_tabelas_origem.ItemIndex]) = -1 then
        lb_tabelas_destino.Items.Add(
          lb_tabelas_origem.Items[
          lb_tabelas_origem.ItemIndex]);
end;

procedure Tfrm_tabelas.btn_delClick(Sender: TObject);
begin
  if lb_tabelas_destino.ItemIndex > -1 then
    lb_tabelas_destino.Items.Delete(
      lb_tabelas_destino.ItemIndex);
end;

procedure Tfrm_tabelas.btn_add_todosClick(
  Sender: TObject);
begin
  lb_tabelas_destino.Items.Clear;
  lb_tabelas_destino.Items :=
    lb_tabelas_origem.Items;
end;

procedure Tfrm_tabelas.btn_del_todosClick(
  Sender: TObject);
begin
  lb_tabelas_destino.Items.Clear;
end;

procedure Tfrm_tabelas.btn_fecharClick(
  Sender: TObject);
begin
  Application.Terminate;
end;
```

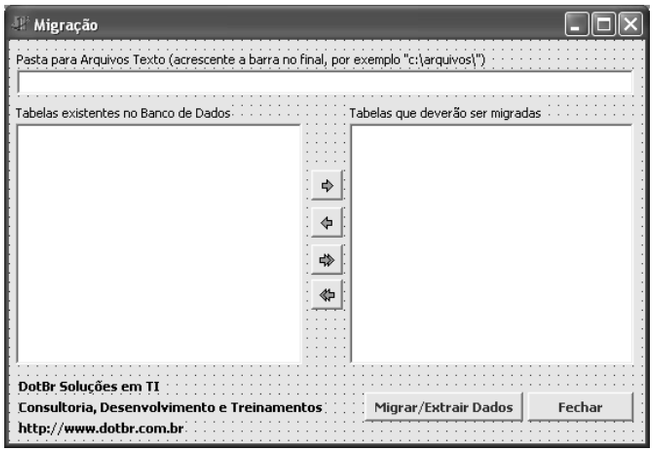


Figura 2. Seleção das tabelas que serão migradas

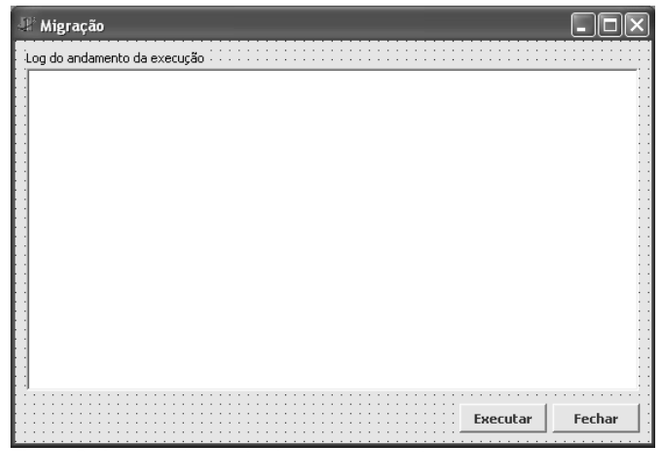


Figura 3. Acompanhamento do processamento

Clube83.indb 14 04.04.07 17:38:03

é necessária em alguns casos, como por exemplo, em campos nulos (precisa colocar a palavra NULL no INSERT) ou campos do tipo *Date* (deve estar com o formato YYYY-MM-DD). O código dessas funções pode ser visto na **Listagem 3**.

Atenção: *AtualizaLog* *deve estar no* ***frm_execucao*** *e os demais procedimentos presentes na* **Listagem 3** *devem estar no Data Module.*

Já o algoritmo da migração dos dados é o seguinte: o sistema seleciona o(s) arquivo(s) texto(s) e, para cada um dos arquivos, abre uma transação e executa os INSERT existentes no arquivo.

Listagem 2. Código para extração de dados das tabelas

```
procedure Tfrm_execucao.ExtraiDados;
var
  fArquivo: TStrings;
  fInsert, fNomeArquivo: string;
  fNrArquivo, fCont1, fCont2, fNrRegistros: integer;

begin
  fArquivo := TStringList.Create;
  AtualizaLog('** INICIANDO EXTRAÇÃO DE REGISTROS **');
  for fCont1:= 0 to
    frm_principal.lb_tabelas_destino.Items.Count - 1 do

  begin
    fNrArquivo := 0;
    fNrRegistros := 0;
    AtualizaLog('Tabela ' +
      frm_principal.lb_tabelas_destino.Items[fCont1] +
      ': obtendo dados');
    dm.qry.SQL.clear;
    dm.qry.SQL.Add('select * from '+
      frm_principal.lb_tabelas_destino.Items[fCont1]);
    dm.qry.open;

    while not dm.qry.Eof do
    begin
      fInsert := 'insert into ' +
        frm_principal.lb_tabelas_destino.Items[
        fCont1] + ' values (';

      for fCont2 := 0 to dm.qry.Fields.Count-1 do
        if fCont2 <> dm.qry.Fields.Count-1 then
          fInsert := fInsert + dm.TransformaDado(
            dm.qry.Fields[fCont2]) + ','
        else
          fInsert := fInsert + dm.TransformaDado(
            dm.qry.Fields[fCont2]) + ');';
      fArquivo.Add(fInsert);
      inc(fNrRegistros);
      dm.qry.Next;

      if ((fNrRegistros/10000) = int(
        fNrRegistros/10000)) or (dm.qry.Eof) then
      begin
        (fArquivo as TStringList).Sort;
        Inc(fNrArquivo);

        fNomeArquivo := UpperCase(
          frm_principal.lb_tabelas_destino.Items[
          fCont1] + '_' + IntToStr(fNrArquivo) +
          '.txt');

        fArquivo.SaveToFile(
          frm_principal.edt_pasta_arquivos.Text +
          '\'+fNomeArquivo);
        AtualizaLog('Arquivo ' + fNomeArquivo +
          ': exportado com sucesso');

        fArquivo.Clear;
      end;

    end;
  end;
  AtualizaLog('*** FIM DA EXTRAÇÃO DE REGISTROS ***');
  AtualizaLog('');
  dm.qry.Close;
  dm.trnsc_origem.Commit;
end;
```

Listagem 3. Atualização do log e transformação de dados

```
procedure Tfrm_execucao.AtualizaLog(pMensagem: string);
begin

  if pMensagem <> '' then
  begin
    log.Lines.Add(DateTimeToStr(now) + ': ' + pMensagem);
    log.Lines.SaveToFile(
      frm_principal.edt_pasta_arquivos.Text + '\log.log');
  end
  else
    log.Lines.Add('');

end;

{ Declarar variável na seção public do DataModule }
fErroExecucao : Boolean;

...
function tdm.MudaCaracter(pString, pCaracterVelho,
  pCaracterNovo: string) : string;
var
  fCont: integer;
begin

  if length(pString) = 0 then
    Result := pString
  else
  begin
    fcont := pos(pCaracterVelho, pString);

    while fcont > 0 do
    begin
      pstring:= copy(pString, 1, fcont-1) +
        pCaracterNovo + copy(pString, fcont + length(
        pCaracterVelho), 3000);
      fCont := pos(pCaracterVelho, pString);
    end;
    Result := pString;
  end;

end;

function Tdm.InverteData(pField: TField): string;
var
  d, m, a: word;
begin
  case pField.DataType of
    ftDate:
      begin
        DecodeDate(pField.AsDateTime, a, m, d);
        Result := IntToStr(a) + '/' + IntToStr(m) +
          '/' + IntToStr(d);
      end;

    ftDateTime:
      begin
        DecodeDate(StrToDate(copy(pField.AsString, 1,
          10)), a, m, d);
        Result := IntToStr(a) + '/' + IntToStr(m) +
          '/' + IntToStr(d) + copy(pField.AsString,
          11, 9);
      end;
  end;
end;

function Tdm.TransformaDado(pField: TField): string;
begin

  if pField.IsNull then
    Result:= 'null'
  else
  begin
    case pField.DataType of
      ftFloat: Result := dm.MudaCaracter(pField.AsString, ',' ,'.');
      ftString:

        begin
          Result := dm.MudaCaracter(pField.AsString, ';', ':');
          Result := dm.MudaCaracter(Result, '/*', '|*');
          Result := dm.MudaCaracter(Result, '--', '- ');
          Result := QuotedStr(dm.MudaCaracter(Result, chr(39),' '));
          { chr(39) = ' (apóstofro) }
        end;

      ftDateTime,ftDate: Result := QuotedStr(dm.InverteData(pField));
      else

    Result := pField.AsString;

    end;

  end;
end;
```


Clube83.indb 15 04.04.07 17:38:04

Se não houver nenhum erro, a transação recebe um *Commit* e o arquivo é renomeado para OK_ANTIGO_NOME, do contrário recebe um *Rollback* e o arquivo é renomeado para ERRO_ANTIGO_NOME. O código da *MigraDados* está na **Listagem 4**.

Listagem 4. Código para migração dos dados

```
procedure Tfrm_execucao.MigraDados;
var
  fNrRegistros, fCont: integer;
  fSearchResult: TSearchRec;
  fArquivo: TStrings;
begin
  fArquivo := TStringList.Create;
  { pega a lista de arquivos na pasta indicada }
  if FindFirst(frm_principal.edt_pasta_arquivos.Text +
    '*.txt', faAnyFile, fSearchResult) = 0 then
  begin
    repeat
      fArquivo.Add(fSearchResult.Name);
    until FindNext(fSearchResult) <> 0;
    FindClose(fSearchResult);
  end;
  dm.trnsc_destino.StartTransaction;
  AtualizaLog('** INICIANDO MIGRAÇÃO DE REGISTROS **');
  for fCont:=0 to fArquivo.Count-1 do
  begin
    AtualizaLog('Migrando arquivo ' + fArquivo[fCont]);
    dm.script.SQL.LoadFromFile(
      frm_principal.edt_pasta_arquivos.Text +
      fArquivo[fCont]);
    dm.fErroExecucao := False;
    dm.script.ExecSQL;
    if not dm.fErroExecucao then
    begin
      dm.script.Transaction.CommitRetaining;
      fNrRegistros := fNrRegistros +
        dm.script.SQL.Count;
      AtualizaLog('Arquivo ' + fArquivo[fCont] +
        ' - SUCESSO');
      RenameFile(fArquivo[fCont],
        frm_principal.edt_pasta_arquivos.Text +
        '\OK_' + fArquivo[fCont])
    end
    else
    begin
      dm.script.Transaction.RollbackRetaining;
      AtualizaLog('Arquivo ' + fArquivo[fCont] + ' - FALHOU');
      RenameFile(fArquivo[fCont],
        frm_principal.edt_pasta_arquivos.Text + '\ERRO_' + fArquivo[fCont])
    end;
  end;
  AtualizaLog('*** FIM DA MIGRAÇÃO DE REGISTROS ***');
  AtualizaLog('');
  dm.trnsc_destino.Commit;
end;
```

Por último, o mais simples dos algoritmos, o que compara o número de registro de cada uma das tabelas (*ComparaDados*). Para tanto, o sistema deve obter o número de registros da tabela no banco de dados original e no novo banco de dados (basta usar o comando *select count(*) from nome_tabela*) e comparar os números.

Se forem diferentes deve adicionar uma linha no *log* com o nome da tabela cujo número de registros não bate. Como esse código é bem mais simples, deixarei a cargo do leitor a sua implementação.

Para finalizar os trabalhos na *unit* do *frm_execucao*, inclua no evento *OnClick* do *btn_executar* uma chamada para *ExtraiDados*, *MigraDados* e *ComparaDados* e uma chamada ao *Close* do *btn_fechar*.

De volta ao *frm_principal*, acrescente uma chamada ao evento *GetTableNames* (traz o nome de todas as tabelas existentes no banco) do *db_origem* no evento *OnCreate* do formulário:

```
dm.db_origem.GetTableNames(lb_tabelas_origem.Items);
```

E no evento *OnClick* do *btn_migrar_extrair* coloque o código:

```
frm_execucao := Tfrm_execucao.Create(self);
try
  frm_execucao.ShowModal;
finally
  frm_execucao.Free;
end;
```

Ganho de desempenho obtido

O trabalho de otimização visava atender principalmente duas coisas: melhorar a importação de arquivos (dados em arquivos textos que são importados para o sistema) e diminuir o tempo de emissão dos relatórios gerenciais. A **Tabela 1** mostra o ganho de desempenho obtido na importação dos arquivos texto.

Tamanho do Arquivo	Tempo Antes (hh:mm)	Tempo Depois (hh:mm)	Redução do Tempo
1.4 MB	02:06	00:34	73,1 %
3.2 MB	03:56	01:17	67,4 %
5.8 MB	06:51	02:41	60,8 %

Tabela 1. Ganho de desempenho na importação dos dados

Com o banco de dados corrompido e sem as mudanças de configuração apontadas no começo do artigo, eram processados aproximadamente três arquivos numa noite, após o trabalho esse número passou para oito.

Já a **Tabela 2** mostra que o ganho de desempenho obtido na emissão de alguns dos relatórios mais usados no sistema foi de aproximadamente 75% (os nomes dos relatórios foram propositalmente ocultados).

Relatório	Tempo Antes (hh:mm:ss)	Tempo Depois (hh:mm:ss)	Redução do Tempo
Relatório 1	00:14:12	00:03:19	76,6 %
Relatório 2	01:38:16	00:26:01	73,5 %
Relatório 3	15:36:19	04:04:37	73,8 %

Tabela 2. Ganho de desempenho na emissão dos relatórios

Conclusão

Muitas vezes, mesmo com o auxílio de ferramentas de reconhecida qualidade, não é possível reparar um banco de dados corrompido. Nesses casos, têm-se duas opções: restaurar um *backup* e perder tudo o que foi adicionado/alterado/excluído depois do *backup* ou migrar todos os dados, registro a registro, como nesse caso. ■

Clube83.indb 16 04.04.07 17:38:04