Lendo DBF em AdvPL — Sem DRIVER ou RDD

Publicado em **04/01/2019**

INTRODUÇÃO

Com o passar do tempo, a utilização de tabelas em formato ISAM, como o DBF e o c-TREE, estão sendo substituídas por uso de tabelas temporárias no Banco de Dados principal das aplicações AdvPL. E, por questões tecnológicas, a utilização do Protheus Server 64 Bits não suporta mais abertura e manutenção de arquivos no formato DBF. Neste post eu compartilho uma forma de ler conteúdo de arquivos DBF em AdvPL, independente do sistema operacional (Windows / Linux) e independente da Build do Application Server (32 ou 64 bits).

DRIVER DBF EM ADVPL

Em breve será lançada uma build nova do Protheus 12 (ERP Microsiga) chamada “Lobo Guará”, para Windows 64 Bits e Linux 64, não devendo mais haver Build 32 bits, e o novo servidor não terá mais suporte ao uso de arquivos DBF.

Muitas vezes, existem integrações com sistemas legados que ainda usam arquivos no formato DBF. Eu mesmo tenho várias modelagens de sistemas antigos — coisas de mais de 20 anos — todas criadas com o driver DBFNTX do Clipper 5.x, campos memo em formato DBT. Após estudar um pouco a estrutura interna do arquivo DBF, e dos campos MEMO (DBT e FPT), resolvi construir uma classe em AdvPL que seja capaz de abrir e ler os dados de um arquivo DBF usando as funções de baixo nível de arquivo do AdvPL (FOpen, FRead,FSeek).

CLASSE ZDBFTABLE

A implementação da classe foi feita partindo da premissa que todas as operações de leitura da tabela sejam feitas através de métodos, que foram publicados com o mesmo nome das funções de baixo nível de arquivo de dados do AdvPL — por exemplo DbSkip(), DBGotop(), FieldGet().

A primeira versão operacional suporta leitura de arquivos nos formatos dBASE III/Foxpro/Foxbase+/Clipper, com campos MEMO em arquivo DBT e/ou FPT. Não há suporte a índices ou filtros, todos os registros são lidos — inclusive os registros deletados logicamente

(ou seja, marcados para deleção). Os dados são lidos na ordem física de registro — ou Recno(). Os métodos de navegação ISAM e de leitura implementados são:

- *DBGoBottom()* — Posiciona no último registro da tabela.
- *DBGoTop()* — Posiciona no primeiro registro da tabela
- *DBSkip(nRecs)* — Avança ou volta um ou mas registros
- *DBGoto(nRecno)* — Posiciona direto em um registro pelo número.

O construtor do objeto — método *New()* — recebe como parâmetro o nome do arquivo DBF no disco a ser aberto. O arquivo é aberto apenas para leitura usando o método *Open()*, que já posiciona a tabela no primeiro registro, e fechado usando o método *Close()*. A recuperação de dados e de status dos registros é feita usando os métodos:

- *FieldGet(nCol)* — Recupera o valor de uma coluna da linha atual
- *FieldPos(cCol)* — Recupera o número da coluna a partir do nome
- *EOF()* — Retorna .T. caso o ponteiro de registros não esteja apontando para um registro. Isto normalmente acontece quando a tabela está vazia ou quando houve a movimentação de ponteiro para próximo registro, e você já estava no último registro da tabela.
- *BOF()* — Retorna .T. caso a ultima movimentação de registro tentou ler antes do primeiro registro da tabela
- *Recno()* — Retorna o número do registro atual posicionado da tabela. Caso ela esteja em EOF(), é retornado o número do último registro mais um.
- *Deleted()* — Retorna .T. caso o registro atualmente posicionado esteja deletado.

Outros métodos para recuperar propriedades da tabela são:

- *LastRec()* e/ou *RecCount()* — Retornam o número do último registro da tabela. Como os registros no DBF são numerados a partir do número 1, ele também representa o número total de registros da tabela.
- *DBStruct()* — Retorna um Array contendo a estrutura do arquivo DBF, com 4 colunas: Nome do campo, Tipo do Campo, Tamanho do campo e quantidade de decimais.
- *Header()* — Retorna o tamanho em bytes do cabeçalho do arquivo DBF
- *RecSize()* — Retorna o tamanho em bytes ocupado no arquivo DBF para uma linha de dados — Equivale a soma do tamanho de todas as colunas mais um (Coluna interna reservada para marcar um registro como deletado.)
- *LUpdate()* — Retorna a data da última atualização de registro do arquivo DBF, gravada no Cabeçalho do arquivo.

FONTE DE EXEMPLO

Sem maiores delongas, segue abaixo o fonte **“LeUmDbf.prw”**, com o exemplo de uso da classe ZDBFTable(). Ela abre uma tabela DBF, mostra informações sobre a tabela e a estrutura de dados, e o primeiro registro (RECNO 1) da tabela no log de console.

```
#include  'protheus.ch'

User Function LeUmDBF()
Local oDBF
```

```
Local nCampos, aStru
Local cFile := '\system\sx3990.dbf'
Local nI

// Cria o objeto para leitura do arquivo
oDBF := ZDBFTABLE():New(cFile)

// Faz a abertura. Em caso de falha,
// aborta a aplicação obtendo os detalhes do erro
// usando o método GetErrorStr()
If !oDBF:Open()
        UserException( oDBF:GetErrorStr() )
Endif

// Mostra no log de console alguns dados sobre o arquivo aberto
conout(replicate('=' ,79))
conout("Database File Name : " + cFile )
conout("Database File Type : " + oDBF:GetDBType()+ " => "+ oDBF:GetDBT
conout("Last Update .......: " + dtoc(oDBF:LUpdate()) )
conout("Record Count ......: " + cValToChar(oDBF:RecCount()))
conout("Header Size .......: " + cValToChar(oDBF:Header()))
conout("Record Length......: " + cValToChar(oDBF:RecSize()))

// Recupera a estrutura da tabela
aStru := oDBF:DbStruct()

// Verifica quantos campos tem a tabela
nCampos := len(aStru)

// Mostra a estrutura de campos no console
conout("")
conout("--- Table Structure --- ")

For nI := 1 to nCampos

        conout("Field ["+aStru[nI][1]+"]"+;          // Nome
          " Type ["+aStru[nI][2]+ "]"+;          // Tipo (CNDLM)
          " Size ["+Str(aStru[nI][3],3)+"]"+;    // Tamanho
          " Dec ["+Str(aStru[nI][4],2)+"]")     // Decimais

Next

// Mostra o primeiro registro da tabela, e detalhes do registro
// Caso a tabela esteja vazia, BOF() e EOF() retornam .T.

conout(replicate('-' ,79))
conout("RECNO() ..... "+cValToChar(oDBF:Recno()) )
conout("BOF() ........ "+cValToChar(oDBF:Bof()) )
conout("EOF() ........ "+cValToChar(oDBF:Eof()) )
conout("DELETED() .... "+cValToChar(oDBF:Deleted()) )
conout("")
For nI := 1 to len(aStru)
        conout(oDBF:FieldName(nI)+" => " +;
```

```
                "["+cValToChar(oDBF:Fieldget(nI))+"]" )
    Next
    conout("")

    // Fecha a tabela
    oDBF:Close()

    // Limpa / Libera o Objeto
    FreeObj(oDBF)

    Return
```

DESEMPENHO

Não é uma API em C/C++, mas o desempenho não ficou nada mal. Usando uma tabela de 44 colunas e 1660 bytes por registro (ou linha), o programa leu sequencialmente em média 5 mil linhas por segundo. Este desempenho pode ficar ate dez vezes mais lento, quando o arquivo a ser lido estiver sendo acessado por um compartilhamento de rede por exemplo — não têm almoço grátis.

QUER USAR ESTA CLASSE ?

*Para usar esta classe, pegue o código fonte atualizado no GITHUB — arquivo **"zDBFTable.prw"** – – e fique a vontade. Apenas mantenha o cabeçalho do fonte com as informações da autoria do código. **Simples assim*** 🤪

Como ela acessa o arquivo diretamente no disco, ela funciona para qualquer Build do Protheus, 32 e 64 Bits, Windows e/ou Linux, e como as funções utilizadas são arrox-com-feijão, esse fonte pode ser usado em versões anteriores do Protheus Server, lançadas a pelo menos nos últimos 10 anos.

CONCLUSÃO

Para uma primeira versão de leitura, eu acho que já dá pra fazer bastante coisa. Ao usar a classe, use sempre o acesso pelos métodos, e não diretamente as propriedades, e também não use os métodos de uso interno — prefixados com "_". Desta forma é possível eu continuar evoluindo esta classe sem impacto nos fontes que a utilizam. O fonte da classe já está no GITHUB, no próximo post nós vamos entrar dentro dos detalhes da implementação do DBF Driver em AdvPL!

Desejo a todos um ótimo uso da classe ZDBFTABLE, e TERABYTES DE SUCESSO !!!

REFERÊNCIAS

- GITHUB – Blog Tudo em AdvPL
- .dbf – Wikipedia

Publicado em **BLOG** por **Júlio Wittwer**. Marque **Link Permanente [https://siga0984.wordpress.com/2019/01/04/lendo-dbf-em-advpl-sem-driver-ou-rdd/]** .

Pingback: Classe ZDBFTABLE – Parte 01 – Introdução | Tudo em AdvPL

Elson de Andrade
em **24/06/2019 às 08:32** disse:

Excelente, tenho muitos fontes antigos que acessam diretamente os DBFs, vou até acrescentar um acesso direto ao valor referente ao campo o que hoje eu faria assim: oDBF:FieldGet(oDBF:FieldPos(‘E3_VEND’)) quando não sei a posição dela.
Eu não entendi sobre a utilização de arquivos de índices anteriormente abertos assim: SET INDEX TO MEUINDICE, Pode me explicar?

★ Curtido por 1 pessoa

Júlio Wittwer
em **25/06/2019 às 22:31** disse:

Opa, perfeitamente ..rs.. Quando você usa uma RDD ou Driver do AdvPL para acesso a dados, o comando SET INDEX TO é traduzido em tempo de compilação para a função DBSetIndex(), usada para abrir um indice. Como essa implementação é oriunda do DBF, se você não abrisse o índice após abrir a tabela, além de não ser possível usar a busca indexada (DBSeek), caso você fizesse uma inclusão de registro ou uma atualização de campos chave de um registro, sem o índice aberto, ele ficava desatualizado, e posteriormente quando você abrisse o índice, uma busca por DBSeek não encontrava a informação.

O Driver de DBF que eu montei não trabalha com índices em disco, mas permite a criação de um índice em memória usando array, tornando possível estas operações 😀

Abraços e grato pela audiência 😀

★ Curtir

Pingback: O que é CODEPAGE e ENCODING – Parte 04 | Tudo em AdvPL

Rodolfo Rosseto
em **12/11/2019 às 14:03** disse:

Eu quero usar esta classe para conseguir fazer as manipulações no DBF até que a gente consiga retirar todos eles dos fontes no Release 25. Tem como incluir algo de forma fácil desta forma? É que temos alguns relatórios em Crystal Report que também vamos migrar, mais queríamos deixar para reescrever após a versão já estar rodando. Grato.

★ Curtir

Júlio Wittwer
em **12/11/2019 às 20:44** disse:

Opa, perfeitamente. A classe para DBF depende de outras classes e funções, que fazem parte do projeto da ZLIB — Basta baixar os fontes e includes da ZLIB — https://github.com/siga0984/zLIB — , compilar no seu projeto e usar. A licença é totalmente livre e permissiva, use/clone/implemente a seu gosto 😀 O Que não dá para fazer é emular em AdvPL um “alias” para a tabela, a classe exige que você acesse os dados e campos pelos métodos implementados 😉

★ Curtir

Rodolfo Rosseto
em **21/11/2019 às 09:26** disse:

Juliao, valeu mesmo. Consegui fazer uma adaptação na rotina para prosseguir com a migração com sua função. Pelo menos até retirarmos completamente o DBF do cliente. Valeu mesmo.

★ Curtido por 1 pessoa

Júlio Wittwer
em **22/11/2019 às 12:52** disse:

Beleza !!!! 😀 😉

★ Curtir

Felipe Rossi
em **26/12/2019 às 16:14** disse:

Júlio, excelente matéria!
Sabe qual a melhor alternativa para substituir o comando Copy to &(cArquivo) via “DBFCDXADS”?
Na versão 64 bits do Application Server essa funcionalidade do copy to para DBF não está mais disponível.
Consigo algo parecido na classe ZDBFTABLE?
Um abraço.

★ Curtido por 1 pessoa

Júlio Wittwer
em **27/12/2019 às 13:33** disse:

Opa, obrigado ! Sim, consegue sim. Basta pegar a estrutura do alias atual em uso usando DBStruct(), criar a tabela DBF usando a ZDBFTABLE, depois fazer um While !eof() no alias, lendo os campos com Fieldget() e acrescentando os novos registros no DBF e atribuindo o conteudo usando os metodos da ZDBF-TABLE 😜

★ Curtir

Felipe Rossi
em **02/01/2020 às 11:22** disse:

Tá feito!
Valeu por compartilhar seu conhecimento!

```
oDBF := ZDBFFILE():New(cArquivo)
oDBF:Create(aEstrutura)
oDBF:Open(.T., .T.)

While SQLTABLE->( !Eof() )
if oDBF:Insert()
for nE := 1 to len(aEstrutura)
oDBF:FieldPut(nE, SQLTABLE->&(aEstrutura[nE][1]))
next
oDBF:Update()
endif

SQLTABLE->( dbSkip() )
EndDo

oDBF:Close()
```

★ Curtido por 1 pessoa

Júlio Wittwer
em **03/01/2020 às 01:39** disse:

Perfeito !! Exatamente isso 😀 (Y)

★ Curtir

José Murilo
em **05/02/2020 às 11:57** disse:

Fala Júlio, bom dia!

Apenas um toque, comecei a testar o seu exemplo, a partir de ontem a tarde, fiz o ajuste para zDBFFILE, mas apanhei um pouco na linha abaixo
aStru := oDBF:DbStruct() //ocorre o erro: Cannot find method ZDBFFILE:DBSTRUCT

Fuçando um pouco mais na zDBFFILE, encontrei a solução, o correto é:
aStru := oDBF:GetStruct()

Cara, é muito massa esse seu blog, muito obrigado por compartilhar sua experiência.

Abraços e tudo de bom.

★ Curtido por 1 pessoa

José Murilo
em **05/02/2020 às 15:07** disse:

Júlio,

Não encontrei como fazer um dbSkip(), para percorrer o dbf lido pelo exemplo que postou, haveria algum método com a mesma função do dbSkip()?

Abraços!

★ Curtido por 1 pessoa

José Murilo
em **05/02/2020 às 17:00** disse:

Júlio,

Favor desconsiderar minha última dúvida, pesquisando mais achei Skip() (dos primórdios do clipper) ao invés de dbSkip(), estava acostumado com este último.

Então usei: oDBF:Skip() e funcionou de boa.

Abraços!

★ Curtido por 1 pessoa

Júlio Wittwer
em **06/02/2020 às 13:33** disse:

Beleza 😃 Manda ver !!!

★ Curtir

Daniela
em **20/02/2020 às 14:37** disse:

Boa tarde Júlio, tenho alguns relatórios customizados no Protheus que obviamente não rodam mais na 25. Eles geravam um DBF e depois explodia em tela a consulta no Excel. Pode me dar uma luz de como faço para converter estes fontes e eles voltarem a funcionar, não manjo muito de programação, mas se puder me dar um norte eu tento aqui. Muito obrigada. Daniela.

★ Curtido por 1 pessoa

Júlio Wittwer
em **14/03/2020 às 18:32** disse:

Olá Daniela, perdoe o atraso na resposta. Dê uma olhada nos posts da classe zDBFFile, voce pode baixar a ZLIB no GITHUB, compilar, e usar a classe para ler e gravar DBF. Ela tem algumas limitações em relação ao DBF suportado pelo AdvPL, mas para integrações e processos exclusivos, funciona legal ! 😃
Mas, conhecimento em programação e AdvPL é necessário para esta tarefa.

★ Curtir

Pingback: Links Uteis – Totvers Advpl

xxxmorgan
em **09/06/2020 às 17:07** disse:

Oi Júlio, tudo bem?

Estou testando a classe que informou no Github e estou tendo como erro “ERROR: Cannot find method NEW – NEW – ZDBFFILE.PRW(153)”.

Estou usando da seguinte forma:
oDBF := ZDBFFILE():New(cOpenDBFD+cArqDBF)

O que posso ter feito de errado? Obrigado!

★ Curtido por 1 pessoa

Júlio Wittwer
em **12/06/2020 às 07:59** disse:

Opa, beleza ?

Então, a classe ZDBFFILE depende da ZISAMFILE e outras classes do projeto ZLIB,. Verifique se voce compilou o projeto ZLIB completo 😃

Abraços

★ Curtido por 1 pessoa

xxxmorgan
em **12/06/2020 às 13:42** disse:

Oi Júlio, obrigado pela resposta!

Pensei que não precisaria e usei apenas esta, com os .ch referidos.

Vou fazer um teste e te retorno!

Nesse meio tempo, sabe o que fiz? Acabou que usei um comando para converter um xls[x] para dbf, mas descobri que podia fazê-lo para csv, o que facilitou um pouco a vida. Ia usar este fonte para importar uma planilha!

Executei o libreoffice em headless e fiz a conversão.

Obrigado mais uma vez pelos conteúdos que compartilha aqui!
Abração!

★ Curtido por 1 pessoa

Júlio Wittwer
em **13/06/2020 às 13:39** disse:

Opa, perfeitamente !!! Em caso de sucesso ou falha, comente aqui no post !!!

Abraços e obrigado pela audiência 😜

★ Curtir