



Criando UDFs para o Firebird/Interbase

Conteúdo

- [Introdução](#)
- [Criando UDFs em Delphi - plataforma Windows](#)
 - Iniciando um novo projeto dentro do Delphi
 - Criando uma nova unit para as suas funções
 - Criando uma rotina de módulo matemático
 - Compilando e usando
 - Sobre strings e datas ?
 - Criando uma rotina de "left"
 - Criando algumas rotinas de data

[Criando UDFs para Linux/Unix](#)

- - Criando um arquiv C
 - Criando uma rotina de módulo matemático
 - Compilando e usando
- [Conclusões](#)

Introdução

Oque é uma UDF ?

Uma função definida pelo usuário (UDF) é meramente uma função escrita em qualquer linguagem de programação capaz de gerar uma biblioteca compartilhada. No Windows, essas bibliotecas são mais conhecidas pelo nome de Dynamic Link Libraries (DLL's).

Porque criar uma UDF ?

Pensando bem, as stored procedures podem fazer muita coisa.

A verdade é que o Interbase não possui um grande número de funções internas. Algumas funções básicas que estão faltando são por exemplo funções de cálculo de módulo, formatação de números de ponto flutuante, funções de manipulação de data e strings, etc...

É sabido que linguagens como o Delphi ou o C podem produzir códigos extremamente rápidos para o cálculo do módulo matemático e para vários outros processamentos na manipulação de data, strings e números de ponto flutuante.

É também sabido que escrever uma UDF é uma tarefa muito fácil; no entanto, o usuário que vai fazê-lo pela primeira vez pode se sentir perdido com alguns dos requerimentos básicos...

Alguns faça e não faça

Antes de começarmos com os exemplos de como escrever UDFs, vamos falar sobre o que voce deve e não deve fazer em uma UDF.



Firebird

Quando voce estiver familiarizado com o processo de escrever uma UDF, irá perceber o quanto o IB é extensível através delas.

Os mecanismo para invocar uma UDF é bem simples, e como voce pode escrever as rotinas usando a linguagem da sua preferência, podendo fazer virtualmente qualquer coisa, certo ?

Bom, sim e não... Algumas coisas que voce não pode fazer com as UDFs : Voce não pode passar NULLs para elas. Da mesma forma, uma UDF não pode retornar um valor NULL. Uma UDF não pode trabalhar com a manipulação de transações sendo assim, informações sobre a transação não podem ser passadas para a UDF e portanto a UDF não pode alterar essas informações e retorna-las ao BD.

De certa forma, uma UDF pode estabelecer uma nova conexão com o BD e iniciar uma outra transação se necessário, mas aí entra o ponto sobre o "faça ou não faça" e não o "pode ou não pode".

Quando se escreve uma UDF, deve-se seguir 2 regras básicas :

- A UDF deve ser uma função simples e **rápida**.
- A UDF não deve tentar alterar diretamente o estado do BD

O que isso significa ?

Bem, uma função que manipula strings, calcula o módulo matemático, faz o tratamento de datas, são exemplos de funções simples e rápidas. São boas candidatas à serem UDFs.

Agora, uma função que se conecta ao BD, insere, apaga, ou atualiza registros é provavelmente uma má idéia. Uma função que rode um programa externo que realiza diversas operações complexas também é provavelmente uma má idéia. Porque ? Porque esses tipos de funções podem fazer com que o BD interrompa o processamento transacional, ou pior ainda, pode prejudicar e muito a performance do seu servidor : Logo que uma UDF é chamada, a thread que chamou a UDF é paralisada até que a UDF retorne.

Lembre-se, é claro, que essas são regras gerais. Pode ser que em um caso particular voce seja obrigado à fazer alguma coisa que é considerada ruim mas que no seu caso específico é especialmente boa.

Vamos ao coração da questão agora...

Criando UDFs em Delphi para plataformas Windows

Criando um projeto no Delphi

1. Inicie um projeto de DLL no Delphi (esse é um tipo especial de projeto, quando voce clicka em File -> New).

Criando uma nova unit para as suas funções

2. Click em File e depois New Unit.

3. Pode ser interessante clicar em Save All nesse ponto...



Criando uma rotina para cálculo do módulo

4. Na nova unit :

5. Declare a rotina na seção da interface:

```
function Modulo(var i, j: Integer): Integer; cdecl; export;
```

6. Implemente a rotina na seção de implementação :

```
function Modulo(var i, j: Integer): Integer;
begin
  if (j = 0) then
    result := -1 // just check the boundary condition, and
                // return a reasonably uninteresting answer.
  else
    result := i mod j;
end;
```

7. No código do projeto recém criado, digite o seguinte logo antes do bloco "begin end.":

```
exports
Modulo;
```

De um Build no projeto

8. Click na opção Build e pronto, voce já tem uma DLL criada.

9. Vou mencionar isso apenas uma vez : O melhor jeito de fazer o Interbase encontrar uma DLL é copia-la no diretório UDF à partir do diretório de instalação do IB, que pode ser por exemplo :

c:\Arquivos de Programas\Borland\InterBase\UDF

10. Para usar a UDF, faça o seguinte : Conecte-se à um Banco de Dados usando o ISQL (ou IB Console)

11. Digite o seguinte :

```
declare external function f_Modulo
integer, integer
returns
integer by value
entry_point 'Modulo' module_name 'dll name minus ".dll";
```

12. De um Commit.

13. Agora teste a função...

```
select f_Modulo(3, 2) from rdb$database
```

Fácil, não foi ?!?!



E sobre strings e datas ?

O que eles tem de especial ? Uma string ou data não são considerados valores escalares no Interbase, portanto um cuidado especial deve ser tomado quando se retorna esse tipo de dado para o IB.

Construindo uma função "Left"

Questões sobre alocação de memória

Se voce possui o IB 5.1 ou menor, digite a seguinte declaração na seção interface da sua unit :

```
function malloc(Bytes: Integer): Pointer; cdecl; external 'msvcrt.dll';
```

Se voce usa o InterBase 5.5 ou superior, então voce não precisa desse código. Ao invés disso, tenha certeza que o arquivo `ib_util.pas` está no path do compilador, e que o arquivo `ib_util.dll` está no path real do sistema operacional.

O melhor modo de se fazer isso é colocar

```
c:\Program Files\InterBase Corp\InterBase\include
```

no library path do Delphi. Depois copie

```
c:\Program Files\InterBase Corp\InterBase\lib\ib_util.dll  
para o diretório system do Windows (tipicamente c:\windows\system). Então coloque o  
ib_util.pas na cláusula uses da seção interface da sua unit.  
uses  
...,  
ib_util;
```

Mas porque tantos detalhes estranhos para tratar a alocação de memória ? Porque eu não posso simplesmente alocar a memória usando `AllocMem` ou coisa parecida ? A resposta mais simples é : Voce não pode, portanto não pergunte ! A resposta mais longa seria que os diferentes compiladores usam seus próprios métodos e algoritmos para gerenciar a alocação de memória que foi lhe dada pelo sistema operacional. Por exemplo, o MSVC gerencia a memória de maneira diferente do Delphi. Adivinhe ? O IB é compilado com o MSVC. Nas versões anteriores à 5.5, era necessário que voce linkasse diretamente a DLL de runtime do MS VC, e nas versões superiores à 5.5, o IB fornece uma chamada para que isso seja feito. Bom, vamos prosseguir com a construção da nossa função de manipulação de strings!

Criando a função

Na seção de Interface da nova unit, digite o seguinte :

```
function Left(sz: PChar; Cnt: Integer): PChar; cdecl; export;
```

Na seção implementation, digite :

```
(* Return the Cnt leftmost characters of sz *)  
function Left(sz: PChar; var Cnt: Integer): PChar;  
var  
    i: Integer;
```



Firebird

```
begin
  if (sz = nil) then
    result := nil
  else begin
    i := 0;
    while ((sz[i] <> #0) and (i < cnt)) do Inc(i);
    result := ib_util_malloc(i+1);
    Move(sz[0], result[0], i);
    result[i] := #0;
  end;
end;
```

No código do projeto, na seção "exports", digite :

```
exports
Modulo,
Left;
```

Agora de um Build novamente no projeto...

Para usar a rotina, vá ao ISQL, conecte-se ao BD que voce já tinha utilizado e digite :

```
declare external function f_Left
cstring(64), integer
returns cstring(64) free_it
entry_point 'Left' module_name 'dll name minus ".dll";
```

E teste isso...

```
select f_Left('Hello', 3) from rdb$database
```

Ainda muito fácil, não ?

Criando rotinas para manipulação de datas

No InterBase 6, existem 3 tipos diferentes de "datas" : Date, Time e TimeStamp. Para aqueles familiarizados com o IB 5.5 e anteriores, o tipo TIMESTAMP do IB 6 é o equivalente ao tipo DATE.

Para decodificar e codificar esses tipos em algo realmente compreensível para o seu programa em Delphi, voce precisa de um pouco mais de informação sobre a API do IB. Coloque na sua Unit o seguinte código :

```
interface
...

type

  TM = record
    tm_sec : integer; // Seconds
    tm_min : integer; // Minutes
    tm_hour : integer; // Hour (0--23)
    tm_mday : integer; // Day of month (1--31)
    tm_mon : integer; // Month (0--11)
    tm_year : integer; // Year (calendar year minus 1900)
```



Firebird

```
tm_wday : integer; // Weekday (0--6) Sunday = 0)
tm_yday : integer; // Day of year (0--365)
tm_isdst : integer; // 0 if daylight savings time is not in effect)
end;
```

```
PTM          = ^TM;
```

```
ISC_TIMESTAMP = record
    timestamp_date : Long;
    timestamp_time : ULong;
end;
```

```
PISC_TIMESTAMP = ^ISC_TIMESTAMP;
```

implementation

...

```
procedure isc_encode_timestamp (tm_date: PTM;
                                ib_date: PISC_TIMESTAMP);
    stdcall; external IBASE_DLL;
```

```
procedure isc_decode_timestamp (ib_date: PISC_TIMESTAMP;
                                tm_date: PTM);
    stdcall; external IBASE_DLL;
```

```
procedure isc_decode_sql_date (var ib_date: Long;
                                tm_date: PTM);
    stdcall; external IBASE_DLL;
```

```
procedure isc_encode_sql_date (tm_date: PTM;
                                var ib_date: Long);
    stdcall; external IBASE_DLL;
```

```
procedure isc_decode_sql_time (var ib_date: ULong;
                                tm_date: PTM);
    stdcall; external IBASE_DLL;
```

```
procedure isc_encode_sql_time (tm_date: PTM;
                                var ib_date: ULong);
    stdcall; external IBASE_DLL;
```

Agora vamos escrever algumas UDFs de manipulação de datas!

Na seção interface da unit recém-criada, digite a seguinte declaração :

```
function Year(var ib_date: Long): Integer; cdecl; export;
function Hour(var ib_time: ULong): Integer; cdecl; export;
```

Na seção implementation, digite :

```
function Year(var ib_date: Long): Integer;
var
    tm_date: TM;
```



Firebird

```
begin
  isc_decode_sql_date(@ib_date, @tm_date);
  result := tm_date.tm_year + 1900;
end;
```

```
function Hour(var ib_time: ULong): Integer;
var
  tm_date: TM;
begin
  isc_decode_sql_time(@ib_time, @tm_date);
  result := tm_date.tm_hour;
end;
```

No código do projeto, na seção "exports", digite :

```
exports
Modulo,
Left,
Year,
Hour;
```

Agora recompila o projeto novamente...

Para usar a rotina, vá ao ISQL, conecte-se ao BD e digite :

```
declare external function f_Year
date
returns integer by value
entry_point 'Year' module_name 'dll name minus ".dll"';
```

```
declare external function f_Hour
time
returns integer by value
entry_point 'Hour' module_name 'dll name minus ".dll"';
```

E teste....

```
select f_Year(cast('7/11/00' as date)) from rdb$database
```

Não tão fácil quanto as anteriores, mais ainda bem simples, certo ?

Criando UDFs nas plataformas Linux/Unix

A maioria dos novatos em Linux que iniciaram sua vida de programação nas plataformas Windows ficarão inicialmente meio tímidos com a noção de como criar UDFs para plataformas Unix/Linux.

O processo não poderia ser mais simples, e em alguns casos eu acho até mais fácil e intuitivo que fazer a mesma coisa no Windows.

Sempre que voce compilar um arquivo em C, ele cria um arquivo de "objeto", que é algo que será estaticamente linkado à algum outro código durante a fase de link, que geralmente produz um arquivo executável.



Firebird

É durante essa fase de link que nós diremos ao compilador C para criar um arquivo de biblioteca compartilhada, que é essencialmente um objeto compartilhado que pode ser linkado à um programa em tempo de execução, e não em tempo de compilação.

No windows, nós chamamos essas bibliotecas de Dynamic Linked Libraries, pois as funções da biblioteca são linkadas ao executável dinamicamente em tempo de execução. É por isso que temos a extensão DLL para esses arquivos.

No Unix/Linux, nós chamamos essas "coisas" de bibliotecas compartilhadas, que são essencialmente objetos compartilhados contendo código que pode ser dinamicamente linkado em tempo de execução. Por isso nós temos geralmente a extensão "so" para esses arquivos.

Voce precisa lembrar que não há nada fundamentalmente diferente entre uma biblioteca compartilhada em Linux e uma DLL no Windows. São a mesma coisa, pelo menos em conceito.

Então, como nós criamos uma biblioteca compartilhada no Linux ?

Criando um arquivo em C

Muito fácil, certo ? Apenas crie um arquivo texto com a extensão .c usando o editor de texto de sua preferência (vi,pico,etc...)

Criando uma rotina de cálculo de módulo

```
int modulo(int *, int *);

int modulo(a, b)
    int *a;
    int *b;
{
    if (*b == 0)
        return -1; // return something suitably stupid.
    else
        return *a % *b;
}
```

Construindo e usando

Na linha de comando

```
gcc -c -O -fpic -fwrapable-strings <your udf>.c
ld -G <your udf>.o -lm -lc -o <your udf>.so
cp <your udf>.so /usr/interbase/udf
```

No ISQL

```
declare external function f_Modulo
integer, integer
returns
integer by value
entry_point 'modulo' module_name 'name of shared library';

commit;

select f_Modulo(3, 2) from rdb$database;
```




Realmente muito fácil !

Conclusão

Wow! Criar UDFs realmente é muito fácil - não tem nada complicado - nem mesmo no Linux.

Portanto podemos tirar as seguintes conclusões sobre o assunto :

É muito fácil. Não há motivo para não criá-las caso você precise delas.

Não se deixe levar. As UDFs podem ser muito poderosas, mas não se deixe levar. Seja objetivo quando decidir o que fazer : Serei melhor servido por uma Stored Procedure ou por uma UDF ?

Para mais exemplos veja a biblioteca FreeUDFLib que contém muitas funções para manipulação de diferentes tipos de dados.

Nota do tradutor : Quando o artigo foi escrito, o Kylix ainda não havia sido lançado. Hoje você pode criar UDFs para o IB for Linux usando o Kylix ou até mesmo o free-pascal.