

Busca fonética

**Integridade
Referencial**

**Formatação
de dados**

Editorial

Esta edição comemora **6 meses** de vida da DB FreeMagazine! Agradecemos a todos os leitores, principalmente aos que estão enviando artigos para publicação na revista!

Também estamos estreando um **novo** padrão de **layout** para a revista, mais leve, o que facilita a leitura, tanto no monitor quando para aqueles que gostam de imprimi-la para ler no papel.

Falando agora sobre o conteúdo desta edição, temos um artigo que mostra uma técnica para **formatar dados** retornados de selects de maneira a apresentá-los de forma mais eficiente diretamente nos relatórios.

Em seguida, trazemos um ótimo artigo sobre **busca fonética** em bancos de dados, que inclui um exemplo de implementação de uma função (udf) com esse propósito para ser usada no Firebird.

O terceiro artigo fala sobre **regras de integridade** em bancos de dados relacionais, uma ótima leitura para aqueles que estão começando agora.

Não poderia deixar de mencionar que este ano teremos **três brasileiros** palestrando na **conferência internacional de Firebird**, a ser realizada em Praga - República Tcheca, de 13 a 15 de Novembro. Dois desses brasileiros são os editores da DB FreeMagazine, ou seja, esse que vos fala e o Luiz Paulo; o terceiro é Maurício Longo, que foi o tradutor no 2º FDD e autor de vários livros!

Aproveitando, comunico que acabo de criar meu **blog** pessoal, que entre outras coisas, falará também sobre Banco de Dados, música, etc. Se tiverem tempo, passem por lá: blog.firebase.com.br.

Um grande abraço a todos, e boa leitura!

Carlos H. Cantu
Setembro/2005

DB Free Magazine

Informações

DB FreeMagazine nº 006 - Ano I
Setembro/2005
Contato geral:
webmaster@dbfreemagazine.com.br

Equipe editorial

Carlos H. Cantu
(cantu@dbfreemagazine.com.br)

Luiz Paulo de Oliveira Santos
(lpaulo@dbfreemagazine.com.br)

Contribuíram nessa edição

Luiz Paulo de Oliveira Santos
Carlos H. Cantu
Mauro Pichiliani
Flavio Yamil Gomez

É proibida a reprodução de qualquer parte do conteúdo dessa publicação sem autorização prévia por escrito.

Dica para melhor visualização

Utilize a resolução **1024x768** pixels e configure o *Acrobat Reader* para **Zoom** de **100%**. Feche todas as abas laterais e esconda as barras de ferramentas, liberando o máximo de área útil na tela, ou simplesmente rode a revista em modo *fullscreen*. Usuários de **Linux**: Recomendamos utilizar o *Acrobat Reader* para Linux.

ANUNCIE NA DB FreeMagazine

Valorize seu produto ou serviço!

anuncios@dbfreemagazine.com.br

Saindo do Forno...

Oracle 10g R2 para Windows

A Oracle disponibilizou a mais nova versão do seu banco de dados para a plataforma Windows, que inclui também o “*Oracle’s Database Extensions for .Net*”, que visa facilitar a vida dos programadores .Net no desenvolvimento de aplicações utilizando os produtos da Oracle.

Fonte: <http://www.computerworld.com/databasetopics/data/software/story/0,10801,104596,00.html>

MySQL 5.0

Acaba de sair o primeiro Release Candidate do MySQL 5.0. Entre as novidades estão:

- * Views
- * Stored Procedures e Stored Functions, usando a sintaxe SQL 2003
- * Triggers (row-level)
- * Server-side cursors (read-only, non-scrolling)

Novos *engines* de armazenamento também foram criados, como o ARCHIVE e o FEDERATED.

Fonte: http://www.mysql.com/news-and-events/news/article_959.html

IB Developer

Foi lançada uma nova revista eletrônica (em inglês) dedicada ao InterBase/Firebird. A IB Developer pode ser baixada gratuitamente no site oficial: <http://www.ibdeveloper.com/>

Novas versões pontuais do PostgreSQL

Foram lançadas novas versões pontuais do PostgreSQL (7.3.11, 7.4.9 e 8.0.4), para corrigir alguns problemas identificados recentemente, entre eles: vários memory leaks, problemas com o VACUUM e melhorias na verificação de páginas de log parcialmente escritas.

Fonte: <http://www.postgresql.org/about/news.396>

Firebird Conference 2005

Foi publicada a grade de palestras da **Conferência Internacional de Firebird**, a ser realizada nos dias 13,14 e 15 de Novembro, em Praga (República Tcheca). Esse ano teremos 3 brasileiros palestrando na conferência.

Mais detalhes em <http://www.firebirdsql.org/index.php?op=konferenz>

Sun de olho nos Bancos de Dados

A Sun está de olho no mercado dos bancos de dados, e atualmente está avaliando o PostgreSQL. Aparentemente a intenção é que no futuro, um banco de dados seja integrado ao sistema operacional.

<http://www.computerworld.com.au/index.php/id;116679278;fp;16;fpid;0>

Borland recusa oferta de compra do Delphi

Para a surpresa de poucos, a Borland recusou a oferta de 150 milhões de dólares feita Robert Coates para adquirir o Delphi e produtos coligados. Coates trabalhou na Borland em 2000 por oito meses.

http://www.cbronline.com/article_news.asp?guid=C7ECD7FF-0E9B-48AD-BC09-FB6E56EB2AEA

Anuncie na DB FreeMagazine!

Atinja diretamente milhares de profissionais da área de programação e banco de dados.

Usando o select para formatar dados em relatórios

Mauro Pichiliani

Alguns relatórios gerados a partir de instruções SELECT permitem a apresentação dos dados para os usuários na forma de grade, onde os valores das colunas são colocados linha a linha. Em determinadas situações, as linhas podem ser apresentadas lado a lado, com o objetivo de facilitar a leitura e visualização. Este artigo apresenta uma maneira de recuperar os dados apresentando dois registros em uma mesma linha do relatório.

Duplicando os dados

Quando temos que apresentar informações na forma de grade, isto é, em linhas e colunas, geralmente utilizamos a instrução SELECT para retornar os dados completos da tabela. Com ele, podemos limitar a quantidade de dados retornados através da cláusula WHERE, a fim de filtrar as informações de maneira que possamos visualizar somente o que nos interessa. Temos ainda o recurso de ordenação dos dados oferecido pela cláusula ORDER BY.

A maioria dos relatórios exige que, após o filtro e a ordenação dos dados, todas as linhas sejam divididas em colunas, com o objetivo de facilitar a leitura do relatório. Como exemplo, vamos trabalhar com a tabela chamada TB_SEPARA, que possui três campos:

ID - contendo uma numeração seqüencial sem falhas;

NOME - representa um atributo texto comum de uma tabela;

DATA - representa uma data comum;

A **Tabela 1** mostra quatro linhas desta tabela.

ID	NOME	DATA
1	A	16/09/2005
2	B	15/09/2005
3	C	17/09/2005
4	D	14/09/2005

Tabela 1. Quatro linhas da tabela TB_SEPARA.

Neste artigo, utilizaremos instruções SELECT baseadas no dialeto **T-SQL** do SQL Server, mas com poucas alterações as instruções aqui apresentadas podem ser portadas para outros bancos de dados, sem grandes dificuldades.

A primeira manipulação que iremos fazer nos dados é duplicá-los fazendo com que o relatório possua seis colunas. Para isso, utilizaremos um *inner join* que relacionará a tabela TB_SEPARA com ela mesma, usando um *alias* chamado **A** e outro *alias* chamado **B**, ambos apontando para a tabela TB_SEPARA.

O SQL Server permite colunas com nomes iguais em uma instrução SELECT, mas modificaremos o nome das novas colunas para facilitar a manipulação delas por uma aplicação. A instrução apresentada na **listagem 1** faz a duplicação das colunas e dos dados:

```

1. -- DUPLICANDO OS DADOS E INCLUINDO
2. -- TRÊS NOVAS COLUNAS
3. SELECT A.ID, A.NOME, A.DATA,
         B.ID AS ID1 , B.NOME AS NOME1,
         B.DATA AS DATA1
4. FROM TB_SEPARA AS A,
5.      TB_SEPARA AS B
6. WHERE A.ID=B.ID

```

Listagem 1. Instrução SELECT que duplica os dados e as colunas.

O resultado da execução da instrução apresentada na **listagem 1** pode ser visto na **tabela 2**.

ID	NOME	DATA	ID1	NOME1	DATA1
1	A	16/09/2005	1	A	16/09/2005
2	B	15/09/2005	2	B	15/09/2005
3	C	17/09/2005	3	C	17/09/2005
4	D	14/09/2005	4	D	14/09/2005

Tabela 2. Duplicação dos dados em novas colunas.

Dividindo os dados

Uma duplicação dos dados como a apresentada na **Tabela 2** não faz muito sentido. Dividiremos os dados de forma que eles não fiquem duplicados, isto é, na primeira linha teremos o valor 1 para a coluna ID, 'A' para NOME e 16/09/2005 para a coluna DATA, seguidos pelo valor 2 para a coluna ID1, 'B' para NOME1 e 15/09/2005 para a coluna DATA1.

Com os dados neste novo formato, estaremos economizando duas linhas no relatório. A **tabela 3** apresenta os dados no formato desejado, considerando apenas os quatro registros da tabela TB_SEPARA.

ID	NOME	DATA	ID1	NOME1	DATA1
1	A	16/09/2005	2	B	15/09/2005
3	C	17/09/2005	4	D	14/09/2005

Tabela 3. Desejável divisão dos dados através das colunas.

Para aplicar esta divisão dos dados e colocá-los neste formato, vamos partir de um caso onde a quantidade de linhas retornadas pelo SELECT seja par. Em seguida, trataremos o caso onde a quantidade de registros retornados seja ímpar.

A idéia utilizada para colocar os dados neste formato é pensar em dividí-los e depois relacioná-los. Analisando os dados da tabela representada pelo *alias A*, podemos perceber que os IDs são ímpares. Matematicamente falando, o resto da divisão por dois de cada um dos IDs do *alias A* deve ser igual 1. De forma análoga, os IDs do *alias B* devem ser pares, ou seja, o resto da divisão por dois destes IDs deve ser igual a zero.

A aplicação do operador **módulo (%)** nos permite facilmente saber se um ID é par ou é ímpar.

Usaremos esse método para distribuir os IDs pares para o alias **A** e os ímpares para o alias **B**.

Até aqui planejamos como iremos separar os dados entre os alias, que serão implementados através de sub-queries na nossa instrução SELECT. Uma possível implementação dessa lógica pode ser vista na **Listagem 2**, no entanto, ela não funciona como esperado.

O problema da instrução apresentada na **Listagem 2** é que o SQL Server, assim como muitos bancos de dados, faz um *cross join* com os dados das tabelas utilizadas na cláusula FROM do SELECT, quando não aplicamos uma regra de *join* entre as tabelas da instrução. O que precisamos fazer agora é implementar uma regra de *join* que relacione corretamente os dados dois alias **A** e **B**.

```
1. SELECT A.ID, A.NOME, A.DATA,
2.    B.ID AS ID1, B.NOME AS NOME1,
   B.DATA AS DATA1
3. FROM
4.    (SELECT * FROM TB_SEPARA
   WHERE ID%2=1 ) AS A,
5.    (SELECT * FROM TB_SEPARA
   WHERE ID%2=0 ) AS B
```

Listagem 2. Implementação incorreta da rotina para separação dos registros

Analisando os dados das colunas ID e ID1 na **Tabela 3**, podemos perceber que o valor da coluna ID1 é sempre o valor da coluna ID somado de um. Esta será a nossa regra de *join*, que pode ser vista na instrução SELECT da **Listagem 3**.

Na cláusula WHERE utilizamos a coluna ID para o alias **B**, pois é somente na lista de colunas da instrução que trocamos o nome da coluna para ID1.

No SQL Server temos duas maneiras de fazer *joins*: podemos utilizar a cláusula **INNER JOIN** ou fazer o *join* como mostrado na **Listagem 3**, relacionando as colunas na cláusula WHERE. Qualquer uma das duas maneiras pode ser utilizada, pois elas não influenciam no resultado final que a instrução gera.

```
1. -- APLICANDO O CRITÉRIO DE JOIN
2. -- NA INSTRUÇÃO SELECT
3. SELECT A.ID, A.NOME, A.DATA,
   B.ID AS ID1, B.NOME AS NOME1,
   B.DATA AS DATA1
4. FROM (SELECT * FROM TB_SEPARA
   WHERE ID%2=1 ) AS A,
5.    (SELECT * FROM TB_SEPARA
   WHERE ID%2=0 ) AS B
6. WHERE A.ID+1=B.ID
```

Listagem 3. Instrução SELECT com o critério de join.

Executando a instrução SELECT da **Listagem 3**, obteremos os dados da maneira que desejamos e que foram mostrados na **Tabela 3**.

E para o caso onde o número de linhas da tabela é ímpar? Neste caso, teremos um registro final para o alias **A** mas não teremos um registro relacionado para o alias **B**. Nessa situação, uma maneira adequada seria apresentar os dados do alias **B** como **NULL**.

A modificação necessária na instrução SELECT é simples: basta utilizar um LEFT OUTER JOIN, ou seja, indicar na instrução SELECT que as linhas do alias **A** que não se relacionarem com as linhas do alias **B** também devem ser incluídas no resultado. A instrução final é apresentada na **Listagem 4**.

```
1. -- UTILIZANDO O LEFT OUTER JOIN
2. -- NA INSTRUÇÃO SELECT
3. SELECT A.ID, A.NOME, A.DATA
4.    , B.ID AS ID1, B.NOME AS NOME1,
   B.DATA AS DATA1
5. FROM
6.    (SELECT * FROM TB_SEPARA
   WHERE ID%2=1 ) AS A,
7.    (SELECT * FROM TB_SEPARA
   WHERE ID%2=0 ) AS B
8. WHERE A.ID+1*#=B.ID
```

Listagem 4. Instrução SELECT que utiliza um LEFT OUTER JOIN.

Notem que o operador ***=** foi implementado para indicar o LEFT OUTER JOIN. Supondo

que a tabela TB_SEPARA tenha cinco linhas, a instrução SELECT da **Listagem 4** irá transformar os dados para o formato exibido na **Tabela 4**.

ID	NOME	DATA	ID1	NOME1	DATA1
1	A	16/09/2005	2	B	15/09/2005
3	C	17/09/2005	4	D	14/09/2005
5	E	18/09/2005	NULL	NULL	NULL

Tabela 4. Divisão da quantidade ímpar de dados.

Conclusão

Este artigo apresentou um método para divisão do resultado de um SELECT em colunas, e da separação dos dados, desde que haja uma coluna na tabela que apresente uma numeração seqüencial sem falhas.

Algumas instruções SELECT foram apresentadas para implementar esta divisão utilizando o dialeto T-SQL, que pode ser facilmente convertido para outros dialetos do SQL de outros bancos de dados.

Outras maneiras de dividir os dados podem ser implementadas, com a divisão dos dados entre colunas, mas isso é assunto para um próximo artigo.

Autor
Mauro Pichiliani
Mini-curriculo
Bacharel em Ciência da Computação, mestrando do ITA (Instituto Tecnológico de Aeronáutica) e MCDDBA. Trabalha há mais de 6 anos utilizando bancos de dados, como o SQL Server e Oracle. É colunista de SQL Server do web site iMasters (http://www.imasters.com.br) e pode ser contatado através do e-mail pichiliani@uol.com.br

Avalie esse artigo

Busca fonética no Firebird

Flavio Yamil Gomez

O presente artigo descreve como realizar buscas fonéticas, ou busca por proximidade sonora, no SGDB Firebird 1.5, utilizando a UDF (User Defined Function) MV_SOUNDEX, que implementa o algoritmo SOUNDEX, utilizado pelo SGDB SQL Server 2000 e idealizado para esse propósito.

Introdução

Existem algumas ocasiões em que queremos buscar informações em um banco de dados, mas não sabemos exatamente a grafia das palavras que informamos para realizar a busca. Isso acontece frequentemente com nome de pessoas, ruas, cidades, etc.

Em sistemas utilizados por um grupo restrito de usuários, como os empresariais, esse tipo de problema não é muito evidenciado, por estarem familiarizados com a grafia das palavras, uma vez que fazem parte do cotidiano de seu trabalho. Porém, em outros casos, onde precisamos disponibilizar algum tipo de pesquisa para usuários leigos, por exemplo, por meio da Internet, a necessidade poderá ser diferente.

Um exemplo clássico poderia ser ilustrado por uma interface web, disponível na grande rede, para que as pessoas possam pesquisar empresas ou profissionais que realizem algum tipo de serviço, ou vendam algum produto específico.

Nesse contexto, a implementação de um sistema de busca de informações utilizando os métodos de busca tradicionais, não teria a eficácia desejada. Se o usuário desejar procurar profissionais de psicologia, por exemplo, poderia tentar realizar a busca digitando a palavra *psicologia* (notem a letra “i” depois do “p”), onde os mecanismos de

busca tradicionais, como o *like*, não conseguiriam selecionar os registros corretos, a menos que as informações estejam escritas de maneira idêntica à solicitada.

Uma alternativa para resolver o problema, é a realização de uma busca por proximidade sonora, ou ainda, busca fonética, onde o sistema de banco de dados é capaz de identificar e selecionar registros realizando buscas mais complexas, utilizando algoritmos que permitem localizar palavras cujas pronúncias sejam semelhantes a pronúncia da palavra solicitada pelo usuário.

Alguns SGDBs (Sistemas Gerenciadores de Bancos de Dados) possuem funções de buscas fonéticas implementadas por padrão, como é o caso do *ORACLE* e do *SQL Server*. O Firebird por sua vez, até a versão 1.5, não possui nenhum mecanismo nativo que permita realizar esse tipo de busca, entretanto, tem a versatilidade de aceitar a implementação de **Funções Definidas pelo Usuário (UDF)**, que agregam funcionalidades ao sistema gerenciador do banco de dados.

A UDF **MV_SOUNDEX** foi implementada visando resolver a deficiência do Firebird no que se refere à realização de buscas fonéticas. A função foi desenvolvida em **Delphi**, e implementa o algoritmo **SOUNDEX**, que é o mesmo algoritmo de busca fonética aplicado no **MS SQL Server 2000**, com algumas adaptações para a fonética do idioma português.

Especificações sobre o funcionamento do algoritmo SOUNDEX

SOUNDEX é um algoritmo que codifica uma palavra tentando representar a sua pronúncia fonética. Desta forma, é possível realizar buscas entre palavras que não necessariamente sejam alfabeticamente idênticas (como no caso das procuras tradicionais), mas sim foneticamente iguais.

O algoritmo SOUNDEX consiste em substituir as consoantes da palavra afetada em números.

A **tabela 1** mostra as letras e seus respectivos códigos. O formato do código resultante da conversão de uma palavra será sempre **LNNN**, onde **L** é a primeira vogal ou consoante da palavra, e **NNN** são três números que representam as três primeiras consoantes diferentes da palavra.

Número	Letras
1	B, F, P, V, W
2	C, Ç, G, J, K, Q, S, X, Z
3	D, T
4	L
5	M, N
6	R

Tabela 1 – Grupo de códigos de letras

A conversão da palavra em seu código correspondente, deverá seguir algumas regras.

Regras sobre o algoritmo SOUNDEX

- As vogais são descartadas, com exceção da primeira;
- As consoantes H e Y são descartadas;
- Se duas ou mais consoantes consecutivas equivalem ao mesmo grupo de conversão, a segunda e restante são descartadas;
- Se a primeira letra da palavra for uma consoante e a segunda pertence ao mesmo grupo de conversão desta, a segunda é descartada.

Alguns exemplos de palavras com suas respectivas codificações são demonstrados na tabela a seguir:

Palavra	Código soundex
Pedro	P360
Vapor	V160
Computador	C513
Supermercado	S165
Soundex	S532
Cooperado	C163
Andar	A536
Osso	O200

Tabela 2 – Exemplos de códigos soundex de palavras

Implementação da UDF MV_SOUNDEX

A UDF MV_SOUNDEX foi implementada por meio de quatro funções:

Função **Mv_soundex** é a função principal, o ponto de entrada. Esta função chama a função **soundex**, que por sua vez chama a função **SoundexPalavra** que chama a função **GrupoSoundex**.

No final do artigo podemos ver o código fonte das funções (em Delphi), com seus trechos e rotinas devidamente comentados.

Instalando a UDF MV_SOUNDEX

Para acrescentar a funcionalidade de busca por proximidade sonora ao Firebird, primeiramente é necessário instalar a UDF MV_SOUNDEX. É importante salientar que esta UDF é totalmente gratuita, e pode ser baixada já compilada na área de downloads do site **FireBase**.

Primeiramente, copie o arquivo *MetaUdf.dll* para a pasta **UDF** da instalação do Firebird no servidor. No windows, por padrão, o caminho é: *C:\Program Files\Firebird\Firebird_1_5\UDF*

Depois que a DLL estiver na pasta UDF, rode o seguinte *script SQL* para adicionar a UDF MV_SOUNDEX:

```
DECLARE EXTERNAL FUNCTION MV_SOUNDEX
CSTRING (1000)
RETURNS CSTRING (1000)
ENTRY_POINT 'Mv_Soundex'
MODULE_NAME 'MetaUdf.dll';
```

Testando a UDF MV_SOUNDEX

Uma vez instalada, já é possível utilizar a UDF. Abra o seu editor SQL preferido e execute o seguinte comando:

```
select Mv_Soundex('testando') from
rdb$database
```

O resultado será o código *soundex* da palavra testando, que neste caso é *T235*.

Neste momento, se tudo correu bem, já é possível utilizar a UDF com qualquer tabela do seu banco de dados, usando a seguinte sintaxe:

```
Select Mv_Soundex(SeuCampo)
from SuaTabela
```

Este comando retornará o código *soundex* do campo *SeuCampo* da tabela *SuaTabela*.

Se o valor do campo contiver mais de uma palavra, a UDF MV_SOUNDEX retornará todos os códigos *soundex* correspondentes às palavras, devidamente concatenados.

Para exemplificar a busca por proximidade fonética, precisaremos de uma tabela, que neste exemplo, terá a seguinte estrutura e dados:

Nome da tabela: CLIENTE

CD_CLIENTE	NM_CLIENTE
1	Pedro da Silva
2	Rosana Gladys Contreras
3	Marcello Caça
4	Flavio Yamil
5	Luci Roka

Tabela 3 – Tabela de exemplo

Se desejarmos localizar o registro da cliente de nome Gladys, mas não sabemos exatamente se o nome é Gladys ou Gladis, poderemos utilizar o seguinte código:

```
Select NM_CLIENTE
from cliente
where Mv_Soundex(NM_CLIENTE) LIKE
'%' || Mv_Soundex('GLADIS') || '%'
```

Para localizar palavras que podem estar no meio da *string*, usamos o operador **like**. No exemplo acima, o código *soundex* do nome inteiro da cliente “*Rosana Gladys Contreras*” é **R250G432C536**, no entanto, desejamos procurar apenas por *Gladis*, cujo valor *soundex* é **G432**.

Melhorando a performance da busca

O tipo de busca demonstrado acima, não é um exemplo que prima pela performance do banco, pois para encontrar o registro solicitado, o SGBD precisará percorrer todos os registros, montar o código *soundex* do campo para depois realizar a comparação. Esta técnica funciona bem para tabelas com poucos registros, mas degrada a performance para tabelas com milhares de registros.

Uma solução para o problema, é a criação de mais um campo na tabela que armazene o valor *soundex* do campo utilizado para a pesquisa, assim, podemos fazer a comparação diretamente no campo que contém o código *soundex*. Desta forma, pode-se utilizar uma *trigger* para atualizar o valor do campo *soundex*, toda vez que o campo principal for alterado.

CD_CLIENTE	NM_CLIENTE	DS_SOUNDEX
1	Pedro da Silva	P360D000S410
2	Rosana Gladys Contreras	R250G432C536
3	Marcello Caça	M624C200
4	Flavio Yamil	F410Y540
5	Luci Roka	L200R200

Tabela 3 – Tabela de exemplo com campo *soundex*

Exemplo de *trigger* para atualizar o campo *DS_SOUNDEX* toda vez que for alterado ou incluído:

```
1. CREATE TRIGGER SET_SOUNDEX FOR CLIENTE
ACTIVE
2. BEFORE INSERT OR UPDATE POSITION 0
3. AS
4. BEGIN
5.     new.ds_soundex =
        mv_soundex(new.nm_cliente);
6. END
```

Considerações finais

A possibilidade de realizar buscas por proximidade sonora estende ainda mais as funciona-

lidades do Firebird, possibilita a criação de programas mais sofisticados, com melhor interação e aumenta a produtividade para os seus usuários, gerando novas oportunidades de negócios para os desenvolvedores de sistemas.

Autor
Flavio Yamil Gomez
Mini-curriculo
Flavio Yamil Gomez é graduado em Ciência da computação pela Universidade do Vale do Itajaí (UNIVALI - SC) e pós-graduado em Gestão Empresarial e Estratégias de Informática pela Universidade Gama Filho (UGF - RJ) em parceria com o Instituto Catarinense de Pós Graduação (ICPG - SC). Atualmente leciona as disciplinas de Programação e Informática básica nos cursos de Sistemas de Informação e Administração na Faculdade do Vale do Itajaí Mirim (ASSEVIM - Brusque - SC - www.assevim.edu.br), a disciplina Banco de dados na WEB no Instituto Catarinense de Pós Graduação (ICPG - Blumenau - SC - www.icpg.com.br) e desenvolve sistemas específicos em Delphi e ASP.NET para pequenas e médias empresas, com bancos de dados SQL Server, ORACLE e Firebird. Pode ser contatado por meio do e-mail flavio@assevim.edu.br

```

1. function Mv_Soundex(Str: PChar): PChar; cdecl;
   export;
2. begin
3.   // Ponto de entrada. Chama a função SOUNDEX e
   // passa como
4.   // parâmetro o texto a ser codificado.
5.   // O texto pode conter várias palavras.
6.   Result := Soundex(Str);
7. end;
8.
9.
10. function soundex(str:PChar):PChar;
11. // Recebe várias palavras como parâmetro, e
   // retorna o código
12. // soundex concatenado.
13. var
14.   // Definição das variáveis de trabalho.
15.   palavra, ret, aux, aux_soundex : string;
16. begin
17.   try
18.     // atribui o texto recebido no parâmetro a
       // variável AUX
19.     aux := str;
20.
21.     // Retira espaços em branco do início e do
       // final.
22.     aux := trim(aux);
23.
24.     // Se não tiver texto nenhum, sai da
       // função.
25.     if aux = '' then
26.     begin
27.       result := '';
28.       exit;
29.     end;
30.
31.     // Atribui "branco" a variavel RET.
32.     // Esta variável será o retorno da função.
33.     ret := '';
34.
35.
36.     // Pega cada palavra do texto passado no
       // parâmetro, e codifica o
37.     // seu soundex. Para isso, chama a função
       // SoundexPalavra.
38.     // Ao final desta rotina, teremos todo o
       // código soundex do texto,
39.     // devidamente concatenado, dentro da
       // variável RET.
40.     while pos(' ',aux) > 0 do
41.     begin
42.       // Pega a primeira palavra
43.       palavra := copy(aux,1,pos(' ',aux));
44.
45.       // Retira a primeira palavra da variável
       // AUX.
46.       aux := copy(aux,pos(' ',aux)+1,
length(aux));
47.
48.       // A variável AUX_SOUNDEX recebe o código
       // soundex da palavra.
49.       aux_soundex := SoundexPalavra(PChar(pala
       vra));
50.
51.       // Concatena o código na variável RET.
52.       ret := ret+aux_soundex;
53.     end;
54.
55.     // Calcula o código soundex da última
       // palavra.
56.     aux_soundex := SoundexPalavra(PChar(aux));
57.     ret := ret+aux_soundex;
58.
59.     // Retorna o código soundex concatenado.
60.     result := PChar(ret);
61.   except
62.     result := '';
63.   end;
64. end;
65.
66.
67. function SoundexPalavra(str:PChar):PChar;
68. // Recebe uma palavra como parâmetro, e retorna
   // o seu código soundex.
69. var
70.   // Definição das variáveis de trabalho.
71.   aux,str2,aux2 : string;
72.   j : integer;
73. begin
74.   try
75.
76.     // A variável STR2 recebe a palavra.
77.     str2 := str;
78.
79.     // Transforma para maiúsculas
80.     str2 := uppercase(str2);
81.
82.     // Se a palavra inicia com caracter
       // especial, sai da
83.     // função e retorna branco.
84.     if ((ord(str2[1]) < 65) Or
       (ord(str2[1]) > 90)) then
85.     begin
86.       result := '';
87.       exit;
88.     end;
89.
90.     // retira as aspas
91.     while Pos('\"', str2) > 0 do
92.       str2[Pos('\"', str2)] := ' ';
93.
94.     // retira espaços duplos
95.     while Pos('  ', str2) > 0 do
96.       str2[Pos('  ', str2)] := ' ';

```

Listagem 1. Listagem do código fonte das funções de soundex (continua...)



Vox on Demand

Automação de discagem e emissão de recados

URA - Unidade Remota de Atendimento

Gravação de ligações telefônicas

Site: <http://www.jobvox.com.br>
E-Mail: SAC@JOBVOX.COM.BR

Pracatuba - São Paulo

```

95.
96. // retira ENTERs
97. while Pos(#13#10, str2) > 0 do
98.     str2[Pos(#13#10, str2)] := '';
99.
100. // Variável AUX recebe o primeiro caracter da palavra.
101. aux := str2[1];
102.
103. // Percorre todos os caracteres da palavra e calcula o código
104. // de cada um, chamando a função GrupoSoundex, armazenando
105. // na variável AUX.
106. for j:=2 to Length(str2) do
107.     begin
108.         aux2 := str2[j];
109.         aux := aux + GrupoSoundex(PChar(aux2));
110.     end;
111.
112. // A variável STR2 recebe o código de todos os caracteres.
113. str2 := aux;
114.
115. // Percorre o código e retira caracteres duplicados.
116. for j:=1 to length(str2) do
117.     begin
118.         if str2[j] = str2[j+1] then
119.             str2 := copy(str2,1,j)+copy(str2,j+2,length(str2));
120.     end;
121.
122. // O código soundex deverá conter uma letra e quatro dígitos.
123. // Esta rotina armazena apenas os quatro primeiros dígitos do código.
124. // Se o código não tiver quatro dígitos, preenche com zeros.
125. if length(str2) > 4 then
126.     str2 := copy(str2,1,4)
127. else if length(str2) < 4 then
128.     while length(str2) < 4 do str2:=str2+'0';
129.
130. // Retorna o código soundex.
131. result := PChar(str2);
132. except
133.     result := '';
134. end;
135.
136. // Retorna o código do grupo do caracter informado no parâmetro.
137. var
138.     cr : string;
139.     grupo : string;
140.
141. begin
142.     try
143.         // Passa para maiúscula
144.         cr := uppercase(caracter);
145.
146.         // Verifica qual é o código do grupo do caracter.
147.         if pos(cr,'BFPVW') > 0 then
148.             grupo := '1'

```

```

149.     else if pos(cr,'CççGJKQSXZ') > 0 then
150.         grupo := '2'
151.     else if pos(cr,'DT') > 0 then
152.         grupo := '3'
153.     else if pos(cr,'L') > 0 then
154.         grupo := '4'
155.     else if pos(cr,'MN') > 0 then
156.         grupo := '5'
157.     else if pos(cr,'R') > 0 then
158.         grupo := '6'
159.     else grupo := '';
160.
161. // Retorna o código do grupo do caracter
162. result := PChar(grupo);
163. except
164.     result := '';
165. end;
166.end;

```

Avalie esse artigo

SQL> SELECT * FROM HOSPEDAGEM WHERE QUALIDADE="INSUPERAVEL";

+-----+
| WWW.BAVS.COM.BR |
+-----+




+ PLANOS


WWW.BAVS.COM.BR
+ INFORMAÇÕES

clientes.com.satisfacao
 Email: info@bavs.com.br
 Atendimento Eletrônico 24hrs
 (19) 3421-0251
 Venhas De-las Ambiente separa:
<http://www.lojas.com.br>

PRO I	PRO III	SEMI D. II
100 Mts espaço em disco Firebird 1.0 MySQL 3.23 PHP 4 Perl 5 CGI Destino SSL Gratuito Configuração Gratuita Manutenção: R\$ 2977	300 Mts espaço em disco Firebird 1.5 MySQL 4 PHP 4 / Perl 5 / CGI JSP (Jrunat) Oracle ASP .NET (Movel Ltd) Configuração Gratuita Manutenção: R\$ 5977	1 Gb espaço em disco Firebird 1.5 MySQL 4 PHP 4 / Perl 5 / CGI JSP (Jrunat) Oracle ASP .NET (Movel Ltd) Configuração Gratuita Manutenção: R\$ 18577

Listagem 1. Listagem do código fonte das funções de soundex (continuação)

Integridade Referencial

Luiz Paulo de O. Santos

Os bancos de dados relacionais disponibilizam facilidades aos desenvolvedores que os tornam mais produtivos, mais confiáveis, rápidos e consistentes no acesso às informações em tabelas. Uma dessas facilidades é o recurso de Integridade Referencial (IR).

Quem desenvolveu aplicativos DOS em Clipper deve lembrar bem de como era complicada e lenta a instrução SET RELATION, que “criava” um relacionamento lógico entre tabelas.

No mundo relacional, a integridade referencial feita através de chaves cuida para que os dados estejam sempre consistentes, evitando, por exemplo, o aparecimento de registros órfãos.

Vejamos os tipos de chave disponíveis:

Chave Primária

Utilizada para identificar um único registro em uma tabela. A chave primária impede que tenhamos mais de um registro com valores iguais nos campos definidos por ela, ou seja, se tornarmos um campo chave primária, não poderemos ter mais de um registro com o mesmo valor nessa coluna. Por exemplo: Em um cadastro de pessoas, seria interessante tornar o campo CPF como chave primária, pois não queremos ter uma mesma pessoa cadastrada duas vezes. NOME é um exemplo de campo que não deve ser utilizado como chave primária, pois podemos ter diversos homônimos (pessoas diferentes com mesmo nome). Além dessa verificação, a definição de uma chave primária cria automaticamente um índice para os campos envolvidos, o que pode gerar ganho de performance para determinados tipos de consulta nessa tabela.

Supondo que temos a seguinte tabela em um banco de dados:

CODIGO	NOME	CIDADE	SALARIO
A0001	José da Silva	Americana	R\$ 2.450,00
A0010	Bárbara Alves	Piracicaba	R\$ 8.310,00
B1201	Manoela Tavares	São Paulo	R\$ 1.890,00

Na tabela acima, a chave primária é a coluna CODIGO, logo, em situação alguma podemos incluir outro código A0001, A0010 ou B1201. O banco irá recusar toda entrada duplicada.

Chave Estrangeira

É um tipo de chave criada basicamente para definir um relacionamento entre registros de tabelas diferente, chamado também de relacionamento pai-filho, ou mestre-detilhe, ou mesmo mestre-escravo. No caso, para cada ocorrência do registro da tabela mestre (pai), poderá existir nenhum, um ou mais de um registro relacionados na tabela detalhe (filha), definidos pela chave estrangeira. Diferente da chave primária, a chave estrangeira aceita duplicação de conteúdo nas colunas envolvidas. Assim como na chave primária, a definição de uma chave estrangeira acarreta na criação de um índice composto pelos campos definidos pela chave, podendo haver ganho de performance em determinadas pesquisas, devido à existência do índice.

Cuidado! - Alguns SGBDs permitem o uso da cláusula UNIQUE no momento da definição de uma chave estrangeira, o que impedirá duplicidade na chave, fazendo com que somente relacionamentos de 1 para 1 sejam possíveis. Seguindo as formas normais (teoria da normalização de tabelas), num caso como esse deveríamos aglutinar essas duas tabelas em uma única tabela.

Supondo que temos a seguinte tabela de vendas, abaixo:

NUMVENDA	CODPESSOA	DATA	VALOR
1	A0001	01/09/2005	R\$ 120,00
2	A0001	02/09/2005	R\$ 30,00
3	A0010	10/09/2005	R\$ 210,00
4	B0010	20/09/2005	R\$ 230,00

Observe que a chave primária da tabela acima é a coluna NUMVENDA, definida por um tipo auto-incremento, onde os valores são gerados automaticamente conforme registros forem sendo inseridos. A coluna em vermelho é uma chave estrangeira, onde claramente podemos observar duplicidade, ou “n” ocorrências da chave no decorrer da tabela. No exemplo, a chave estrangeira estaria ligando o campo CODPESSOA da tabela de vendas ao campo CODIGO de uma outra tabela de PESSOAS, através de uma relação de 1 (PESSOAS) para “n” (VENDAS).

No caso de alterações na tabela “pai”, as tabelas relacionadas sofrerão algum tipo de ação para garantir a consistência do relacionamento. As ações sempre partem da tabela “pai” para a(s) tabela(s) filha(s), sendo elas:

1. Nenhuma ação.

Nessa situação, quando a tabela “pai” (1) sofrer alterações ou exclusões, nenhuma ação automática será realizada em relação às tabelas “filhas”. Se você não fizer um tratamento manual para tornar os dados consistentes, um erro será retornado, pois estaria gerando registros órfãos.

2. Cascade

Nesse caso, sempre que a tabela pai sofre exclusões ou alterações, essas mudanças são repassadas à(s) tabela(s) filha(s). Se o campo relacionado da tabela “pai” foi alterado, automaticamente o campo relacionado nas tabelas filhas será alterado para o mesmo valor, sem a necessidade de implementação de nenhum código adicional. Se algum

registro da tabela “pai” for excluído, os registros relacionados a ele nas tabelas “filhas” também serão removidos.

3. Atribuir NULL

Caso a tabela “pai” sofra alterações ou exclusões, os campos relacionados da chave estrangeira nas tabelas filhas assumirão o valor NULL, ficando “jogados” na tabela, sem estarem relacionados com qualquer registro “pai”. Vale lembrar que NULL não é valor, e sim um estado indefinido. Veja mais a frente o item 5 de “Dicas de otimização de chaves”.

4. Assumir o default

Nesse caso, com a alteração ou exclusão dos registros “pai”, os campos relacionados nas tabelas filhas assumirão um valor default.

Observação - *Tanto as chaves primárias como as estrangeiras baseiam-se em índices, ou seja, sua implementação torna-se impossível sem o uso destes. Índices são estruturas organizadas de forma crescente ou decrescente, e que permitem encontrar registros em uma tabela rapidamente, sem precisar percorrê-los um a um. Além disso, os índices podem ajudar na ordenação do resultado das pesquisas (selects), poupando trabalho para o SGBD.*

A maioria dos SGBDs atuais possui um *engine* de pré-processamento de instruções SQL (otimizador), que analisa o código SQL e otimiza sua execução, verificando se é interessante ou não utilizar determinados

índices para a realização da consulta. Em algumas situações específicas, o otimizador determina que uma varredura seqüencial dos registros é mais rápida do que a utilização de um índice. Alguns SGBDs permitem que o desenvolvedor pré-determine o “plano” de consulta utilizado para uma determinada pesquisa, pulando assim a etapa de otimização.

Dicas de otimização de chaves

É notório e do conhecimento de qualquer desenvolvedor que um projeto mal-feito geralmente gera re-trabalho, e no pior dos casos, precisa ser refeito completamente. Logo, deve-se pensar no que fazer (e como fazer), antes de começar a implementar um sistema.

Abaixo citarei algumas dicas que podem acelerar o desenvolvimento de um projeto:

1 Use chaves com campos pequenos

Principalmente na *chave primária*, que tem uma importância muito grande nos bancos de dados relacionais. Quanto menor for o tamanho do campo, mais rápido será o acesso à chave.

2 Evite campos com tamanhos variáveis

Campos de tamanho fixo são mais indicados para serem chaves primárias. Na maioria dos bancos de dados, o ganho de performance no uso de campos de tamanho fixo é significativo e sensivelmente notado pelos usuários.

3 Evitar excesso de índices

A indexação acelera as buscas, porém, o excesso de índices definidos em tabelas que sofrem muitas inserções (como tabelas de *logs* ou de históricos diversos), exclusões ou mesmo alterações do conteúdo de campos

que são chaves, poderá gerar perda de performance, pois a estrutura de todos os índices será atualizada a cada alteração sofrida na tabela. Portanto, não saia por aí criando índices a torto e a direito. Estude caso a caso onde um índice é aconselhável ou não.

4 Evitar, sempre que possível, uso de chaves compostas

Sempre que empregamos chaves compostas (dois ou mais campos), o banco necessita de mais tempo de CPU para processar essas informações, principalmente se os campos forem de conteúdo variável, como proposto no item 2. Logo, apesar de ser um recurso bastante interessante e prático para o desenvolvedor, deve ser usado com certo cuidado, principalmente em tabelas onde o conteúdo dos campos que compõe a chave sofre alterações constantemente, e necessitam ser atualizados em *cascade*.

5 Atribuir NOT NULL para campos que serão empregados como chaves

Deve-se atribuir NOT NULL para todos os campos que serão empregados como chave. Isso é obrigatório no caso das chaves primárias. No caso das chaves estrangeiras, tome cuidado principalmente se tiver utilizando a ação mencionada no item 4 mais acima (Atribuir NULL).

Definindo chaves em SQL

As instruções para criação de chaves primárias e secundárias são básicas em todos os bancos de dados, sofrendo pequenas mudanças de sintaxe de banco para banco. A seguir citaremos como proceder para sua criação, utilizando uma sintaxe tradicional:

Chaves Primárias:

As chaves primárias são criadas com a instrução:

```
1. PRIMARY KEY (campo, [...])
```

Exemplo:

```
1. CREATE TABLE TAB01 (  
2.     col_1 integer not null,  
3.     col_2 integer not null,  
4.     col_3 varchar(2000),  
5.     col_4 varchar(500),  
6.     PRIMARY KEY (col_1, col_2)  
7. );
```

A chave primária pode ser montada no instante da criação da tabela (CREATE TABLE) ou incluída depois, com a tabela já existente, através do comando ALTER TABLE.

Chaves estrangeiras

As chaves estrangeiras são criadas com a instrução:

```
1. FOREIGN KEY (campo, [...])  
2. REFERENCES <tabela> (campo, [...])
```

Exemplo:

```
1. CREATE TABLE TAB02 (  
2.     col_a integer,  
3.     col_b integer,  
4.     col_c varchar(1000),  
5.     FOREIGN KEY (col_a, col_b)  
6.     REFERENCES TAB01 (col_1, col_2)  
7. );
```

A chave estrangeira pode ser montada no instante da criação da tabela (CREATE TABLE), ou incluída depois com a tabela já existente através do ALTER TABLE.

Conclusão

Espero que esse artigo tenha deixado claro para o leitor a importância da integridade referencial, bem como da sua correta implementação.

Um complemento ao assunto abordado nesse artigo seriam as formas de normalização, mas deixaremos isso para uma outra oportunidade.

Autor:

Luiz Paulo de Oliveira Santos

Mini-curriculo

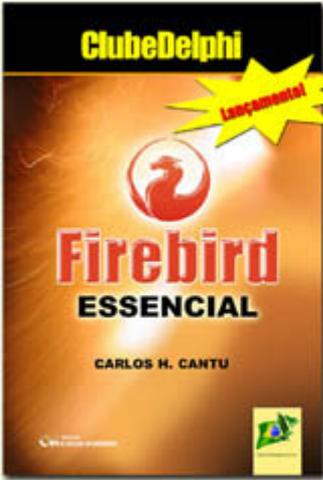
Luiz Paulo de Oliveira Santos é formado em Tecnologia de Processamento de Dados, especialista em Análise de Sistemas e Redes de Computadores. É analista de suporte de redes na Universidade Metodista de Piracicaba e diretor da JobVox Sistemas Informatizados. lpaulo@jobvox.com.br

Avalie esse artigo



Precisando de cursos ou treinamentos de Firebird?

A FireBase oferece cursos/treinamentos de Firebird ministrados dentro da sua empresa. Os cursos são ministrados por **Carlos H. Cantu**, um dos maiores evangelistas do Firebird no Brasil, autor do livro Firebird Essencial. Mais informações pelo email cursos@firebase.com.br



Firebird Essencial

Primeiro livro brasileiro que trata especificamente dos recursos do SGBD Firebird (versões 1.0 e 1.5). O autor reuniu no livro todo o material produzido por ele para as revistas ClubeDelphi e SQLMagazine. Os artigos foram revisados, atualizados e muitos deles complementados, de forma a proporcionar ao leitor uma fonte de informação rica, atualizada e confiável. Um capítulo mérito sobre a criação de UDFs foi escrito exclusivamente para o livro.

Você aprenderá a instalar o SGBD, criar procedures, catálogos em CDROM, criar backups, gerenciar usuários, utilizar campos BLOB de forma adequada, identificar os tipos de dados disponíveis no Firebird, e muito mais!

Verifique o sumário do livro em www.firebase.com.br/fb/livro/fbessencial

www.firebase.com.br

Calendário de Eventos

Data	Evento	Site
03 à 07 de Outubro	20. Simpósio Brasileiro de Banco de Dados Uberlândia - MG	http://www.sbbd-sbes2005.ufu.br/imagens/ChamadaSBBD.pdf
13 à 15 de Outubro	III Encontro de software livre do Amazonas Manaus - AM	http://encontro.comunidadesol.org
10, 11 e 12 de Novembro	2º Encontro Nacional de Usuários MapServer Itajaí-SC	http://www.webmapit.com.br/mapserver/encontro2005/
3,4,5 de Novembro	Conisli – Congresso Internacional de Software Livre São Paulo - SP	http://www.eventosucesusp.org.br/conisli/
13 a 15 de Novembro	Conferência Internacional de Firebird	http://www.firebirdsql.org/index.php?op=konferenz
19 e 20 de Novembro	Latinware Mercosul Curitiba-PR	http://www.softwarelivreparana.org.br
1 e 2 de Dezembro	III Congresso Catarinense e Software Livre Florianópolis	http://www.softwarelivresc.org.br

Se você sabe de algum evento focado em Banco de Dados ou em Software Livre que não esteja listado aqui, envie-nos um email com os dados do evento para que possamos incluí-lo na próxima edição e no calendário do site.



Revista Active Delphi
Assine Já!

- Revista mensal e distribuída em todo Brasil
- Melhor Custo X Benefício
- Ótima referência de pesquisa e estudo
- Apoio da Borland e outras grandes empresas
- 100% programação Delphi
- Profissionais qualificados e certificados escrevem seus artigos

www.activedelphi.com.br/assinatura.php
www.activedelphi.com.br - Tel : (16) 3024-8713



DVD Duplo

Esse DVD duplo contém as 10 palestras apresentadas no 2º Firebird Developers Day, realizado no dia 16 de Julho de 2005, em Piracicaba - SP.

O lucro obtido com a venda do DVD será enviado para a Firebird Foundation, a fim de contribuir com o desenvolvimento do Firebird.

www.FireBase.com.br