DBFree MAGAZINE

Edição #05

Agosto 2005

MySQL®
Iniciando no MySQL

Procedures, Triggers e views

Acessando o PostgreSQL via ADOdb e PHP

Review DB Manager

Foto: vivifrost

Editorial

Olá!

Bem vindos a quinta edição da DB FreeMagazine!

Essa edição traz novamente artigos dos 3 principais SGBDs gratuitos: Firebird, MySQL e PostgreSQL, além de um *review* da ferramenta de administração de dados "brasuca" DBManager!

Todos os artigos foram escritos por brasileiros, o que nos deixa com muito orgulho, pois mostra que não é preciso recorrer ao "exterior" para ter material de qualidade para ser publicado.

Gostaria de fazer uma **convocação geral** para que a comunidade continue enviando artigos para serem publicados! Recebi dezenas de emails de pessoas que estão dispostas a contribuir, mas não basta estar disposto - tem que por a **mão na massa**! :-) Lembrando que a revista é gratuita, mas não se limita a falar de bancos gratuitos - ou seja, aqui tem espaço pra todo mundo, basta o assunto ser interessante e dentro da área de Banco de Dados!

Gostaria de lembrá-los que o site da DB FreeMagazine é dinamicamente atualizado com base em RSS de diversos outros sites. Consultando diariamente nosso portal , você fica atualizado com as últimas notícias da área! Para isso, basta acessar a opção "Notícias Gerais" no menu principal - não é preciso fazer login.

Boa leitura, e bom divertimento!

Carlos H. Cantu
Editor DB FreeMagazine

Informações

DB FreeMagazine nº 005 - Ano I
Agosto/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
Henderson Macedo
Marcelo Santos Araújo

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*.

Saindo do Forno...

Oracle 10g R2

O release 2 do Oracle 10g já está disponível para Linux x86, Solaris, AIX, e HP-UX. Esse release promete aumentar a eficiência e reduzir o custo do gerenciamento de informação.

Fonte: http://www.oracle.com/technology/software/products/database/oracle10g/index.html

Uso de BDs Open Source cresce

Segundo artigo da ComputerWorld, grandes empresas em todo o mundo estão de olho nos bancos de dados open source para substituir os bancos comerciais e reduzir custos.

Fonte:http://www.computerworld.com/databasetopics/data/software/story/0,10801,104278,00.html

HP oferece suporte oficial ao MySQL

A HP passou a oferecer suporte ao MySQL diretamente para seus consumidores através da MySQL Network. Os clientes podem selecionar entre 3 níveis de suporte, com valores e benefícios diferenciados.

Fonte: http://www.computerworld.com/databasetopics/data/software/story/0,10801,104007,00.html

Novell também oferecerá suporte ao MySQL

A Novell parece estar se juntando as inúmeras outras empresas que passaram a oferecer suporte oficial ao banco de dados MySQL. A Novell já possuia uma parceria com a MySQL, que está sendo expandida.

Fonte: http://www.computerworld.com/databasetopics/data/software/story/0,10801,103803,00.html

Novo BD baseado em PostgreSQL

A EntrepriseDB acaba de lançar um "novo" banco de dados, o EnterpriseDB 2005, baseado no PostgreSQL. O banco é gratuito para uso em desenvolvimento e distribuição em pequena quantidade, e pode ter um custo de até US$ 5.000 por CPU anualmente (com suporte incluso).

Fonte:http://www.computerworld.com/databasetopics/data/software/story/0,10801,103798,00.html

Nova pesquisa da Evans Data

A empresa Evans Data está realizando mais uma vez uma pesquisa mundial dedicada a bancos de dados. No ano passado o Firebird teve grande destaque no resultado final da pesquisa. Não deixem de respondê-la!

Fonte: http://www.evansdata.com/EDC_DB_2005_2_Start.html

DVD do 2º Firebird Developers Day

A FireBase está vendendo o **DVD duplo** do 2º FDD, contendo as 10 palestras apresentadas no evento. É uma ótima oportunidade para quem não pode estar presente no evento se atualizar com o conteúdo apresentado.

Fonte:http://www.FireBase.com.br

Firebird 1.5.3 RC1

Enquanto a equipe trabalha na finalização do Firebird 2.0, uma nova versão do Firebird 1.5, chamada de 1.5.3 Release Candidate 1, foi disponibilizad. Ela corrige vários bugs. trazendo algumas das correções que foram feitas no FB 2.0

Fonte: http://www.firebirdsql.org

View, Stored Procedure e Triggers no Firebird

Luciano G. Carvalho
Mariângela F. F. Molina

A simples utilização de instruções SELECT, INSERT, UPDATE e DELETE para a criação de expressões em SQL na manipulação de dados é suficiente para gerenciar pequenas bases de dados, pois a complexidade de elaboração das mesmas é baixa, e quando necessário, podem ser reescritas rapidamente.

Quando grandes bases de dados são construídas, as expressões em SQL necessárias para gerenciá-las são geralmente mais complexas, sendo inviável criá-las toda vez que é necessário fazer uma consulta. Além disso, geralmente há a necessidade de se introduzir no banco de dados, regras de negócios ou rotinas de manutenção de dados. Assim, a utilização de recursos mais sofisticados, tais como VIEW, STORED PROCEDURE e TRIGGER, é justificada, auxiliando de forma eficiente o gerenciamento de tais bases.

View

Uma VIEW (visão) é uma espécie de relação (*tabela*) virtual que é derivada de uma ou mais relações reais de uma base de dados. A estrutura básica para a criação de uma VIEW é:

```
create view <nome da view> as <ex-
pressão da consulta>
```

As visões são utilizadas para vários propósitos, alguns deles são:

- Exibição de dados correlacionados (armazenados em várias relações) em uma só relação, não alterando a estrutura atual do banco de dados;
- Ocultação de dados importantes, já que uma visão pode conter apenas um subconjunto de atributos de uma dada relação, e o direito de acesso pode ser dado à visão, e não diretamente à relação;
- Simplificação de consultas complexas através da criação de visões intermediárias, e da utilização das mesmas na expressão de consulta;
- Permite que os mesmos dados sejam organizados de forma diferente, de acordo com o interesse de cada usuário.

Stored Procedure

Um STORED PROCUDURE (procedimento armazenado) é um código armazenado no Banco de Dados, composto por comandos seqüenciais em SQL. Além de suportar estruturas de repetição (*loops*) e condicionais, é possível definir parâmetros de entrada, saída e variáveis locais.

A estrutura básica de um STORED PROCEDURE é:

```
create procedure <nome do stored pro-
cedure> [( <parâmetros de entrada> )]
[returns <parâmetros de saída>]
as
[<declaração de variáveis>]
begin
<operações do stored procedure>
end
```

A execução de um procedimento é feita da seguinte forma:

```
execute procedure <nome do stored
procedure>
```

As principais vantagens dos procedimentos armazenados são:

- Maior segurança na manipulação dos dados, pois impossibilitam que consultas arbitrárias sejam executadas pelos usuários;
- A execução de procedimentos geralmente é mais rápida, pois os comandos são executados diretamente no servidor, sem tráfego de dados pela rede;
- É possível executar vários comandos SQL a partir da execução de um único procedimento.

Trigger

Um TRIGGER (gatilho) é um procedimento associado a uma relação do banco de dados,

que é disparado automaticamente quando um determinado evento ocorre na mesma. A estrutura básica para a criação de um TRIGGER é:

```
create trigger <nome do trigger> for
<nome da relação>
[active | inactive]
{before | after}
{insert|update|delete} [position <or-
dem de execução>]
as <expressão a ser executada>
begin
end
```

Os gatilhos, além de apresentarem as vantagens dos procedimentos, são executados sem a interferência dos usuários, isto é, a partir da ocorrência de um dado evento na relação, o trigger é disparado pelo próprio servidor. Um exemplo de aplicação desse recurso é o controle de integridade manual do banco de dados.

Exemplo de Implementação de VIEW, STORED PROCEDURE e TRIGGER

Para exemplificar a criação de VIEW, STORED PROCEDURE e TRIGGER, será usado o Sistema Gerenciador de Banco de Dados (SGBD) **FireBird,** e um exemplo clássico de banco de dados usado no livro *"Introdução a Sistemas de Banco de Dados" (Date, 2000)*, mostrado na **figura 1**.

FORNECEDORES

FOR_ID	FOR_NOME	FOR_STATUS	FOR_CIDADE
1	Smith	20	Londres
2	Jones	10	Paris
3	Blake	30	Paris
4	Clark	20	Londres
5	Adams	30	Atenas

PEÇAS

PEC_ID	PEC_NOME	PEC_COR	PEC_PESO	PEC_CIDADE
1	Porca	Vermelho	12	Londres
2	Pino	Verde	17	Paris
3	Parafuso	Azul	17	Roma
4	Parafuso	Vermelho	14	Londres
5	Came	Azul	12	Paris
6	Tubo	Vermelho	19	Londres

FORNECIMENTO

FOR_ID	PEC_ID	QUANTIDADE
1	1	300
1	2	200
1	3	400
1	4	200
1	5	100
1	6	100
2	1	300
2	2	400
3	2	200
4	2	200
4	4	300
4	5	400

Figura 1. Relações do Banco de Dados Utilizado como Exemplo.

Supondo que uma consulta freqüente ao banco de dados seja listar quais peças foram fornecidas por cada um dos fornecedores cadastrados, essa consulta seria uma forte candidata a ser implementada a partir de uma VIEW. A **figura 2** mostra o comando para criação da VIEW e o resultado retornado por ela.

Um bom exemplo de utilização de STORED PROCEDURES é o de efetuar algum tipo de cálculo baseado em parâmetros, como, por exemplo, retornar a quantidade fornecida de uma determinada peça. A construção do referido procedimento e o resultado apresentado para a peça "parafuso" pode ser visto na **figura 3**.

Um recurso exclusivo do FireBird/InterBase é a possibilidade de utilizar Stored Procedures como fonte de dados para consultas em SQL – são as "Stored Procedures *Selecionáveis*".

A estrutura típica de uma consulta envolvendo Stored Procedures Selecionáveis é:

```
select <atributos> from <stored pro-
cedure [parametros]> where <condição>
```

Com relação aos TRIGGERS, uma utilização muito comum é o controle de integridade manual do banco de dados, por exemplo, quando se deseja que a exclusão de um determinado fornecedor implique na exclusão das informações de fornecimento referentes ao mesmo. A **figura 4** mostra a construção do gatilho citado, bem como o estado anterior e posterior da relação Fornecimento para o Fornecedor "Smith" ("FOR_ID" igual a "1").

```
create view vw_pecas_fornecedores ( PECA, FORNECEDOR ) as
select distinct pecas.pec_nome, fornecedores.for_nome
from  pecas, fornecedores, fornecimento
where pecas.pec_id = fornecimento.pec_id and
      fornecedores.for_id = fornecimento.for_id;
```

PECA	FORNECEDOR
Came	Clark
Came	Smith
Parafuso	Clark
Parafuso	Smith
Pino	Blake
Pino	Clark
Pino	Jones
Pino	Smith
Porca	Jones
Porca	Smith
Tubo	Smith

Figura 2. Construção e conteúdo de uma VIEW.

```
CREATE TRIGGER TG_EXCLUSAO_FORNECIMENTO FOR FORNECEDORES
ACTIVE BEFORE DELETE POSITION 0
AS
begin
  delete from fornecimento
  where  fornecimento.for_id = fornecedores.for_id;
end
```

FOR_ID	PEC_ID	QUANTIDADE
1	1	300
1	2	200
1	3	400
1	4	200
1	5	100
1	6	100
2	1	300
2	2	400
3	2	200
4	2	200
4	4	300
4	5	400

FOR_ID	PEC_ID	QUANTIDADE
2	1	300
2	2	400
3	2	200
4	2	200
4	4	300
4	5	400

Figura 4. Construção de um TRIGGER e evolução da relação que foi alvo do mesmo.

```
CREATE PROCEDURE SP_CALCULA_QTD_PECA ( PECA VARCHAR(20) )
RETURNS ( RESULTADO INTEGER )
AS
begin
  select sum( fornecimento.quantidade )
  from  fornecimento, pecas
  where (fornecimento.pec_id = pecas.pec_id and
        pecas.pec_nome = :peca) into :resultado;
  suspend;
end
```

RESULTADO
900

Figura 3. Construção e exemplo de resultado de uma STORED PROCEDURE.

Conclusão

O uso de estruturas mais complexas tais como VIEWS, STORED PROCEDURES e TRIGGERS tornam o gerenciamento dos dados mais eficiente, simplificando consultas e alterações, automatizando tarefas e evitando que o administrador do banco de dados desperdice tempo em recriar comandos.

A criação das estruturas citadas neste artigo pode ser facilitada através da utilização de uma interface gráfica, presente em aplicativos como o IBOConsole e IBExpert. Uma lista dessas ferramentas pode ser obtida na área de downloads do site www.firebase.com.br.

Avalie esse artigo

ÓTIMO	BOM	RUIM

Autores:

Luciano G. Carvalho
lgclive@yahoo.com.br
Mariângela F. F. Molina
Molina_mary@yahoo.com.br

Mini-curriculo

Bacharéis em Ciência da Computação pela Universidade Federal de Itajubá e mestrandos em Engenharia Eletrônica e Computação pelo Instituto Tecnológico de Aeronáutica – ITA.

Iniciando com o MySQL

Henderson Macedo

Esse artigo visa introduzir novos usuários ao MySQL. Discutiremos a instalação do MySQL, os tipos de dados, definição de parâmetros de segurança, e mostrarei como realizar importação e exportação de dados com o banco.

Obtendo o MySQL

O MySQL pode ser obtido através do site: http://www.mysql.com/downloads/

Instalando o MySQL no Windows

Para instalar o MySQL no Windows, é necessário que seja Windows 95 ou Superior e que tenha os drivers de rede para TCP/IP instalado.

Execute o arquivo **setup.exe** e siga os passos para conclusão.

Instalando o MySQL no Linux

É aconselhável que seja instalado a partir de pacotes rpm. Muitas distribuições Linux já vêm com o pacote do MySQL e que pode já estar instalado. Caso não esteja, execute os seguintes comandos no terminal:

```
rpm -ivh mysql-3.23.46-i386.rpm mysql-client-3.23.46-i386.rpm
```

Inicializando o MySQL

Para iniciar o MySQL no Windows, execute o programa *mysqld.exe* que encontra--se no diretório *bin* que está no diretório de instalação do MySQL.

Para iniciar o MySQL no Linux, digite os comandos abaixo:

```
service mysql start
```

A partir de agora, o MySQl estará ouvindo pela porta **3306 TCP**. Para utilizarmos o MySQL Client, usaremos o Prompt do MSDOS no Windows e o Terminal no Linux. No Windows, posicione-se no diretório de executáveis do mysql, por exemplo, em `c:\mysql\bin`. No Linux, o mysql já estará no path. Digite **mysql**. Agora estamos rodando o MySQL Client e se tudo ocorreu corretamente, aparecerá a seguinte tela:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 3.23.46-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

Mysql>
```

Experimente digitar “show databases;” sem aspas no prompt do mysql e obterá o seguinte resultado:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 3.23.46-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show databases;
+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+
3 rows in set (0.00 sec)

mysql>
```

Verifique que cada comando no mysql deve terminar com **;** ou **\g**. No exemplo acima, o comando *show databases* gera uma lista de todos os bancos de dados existentes. O BD *mysql* é onde ficam armazenadas todas as informações relativas à segurança, permissões de acesso, senhas, usuários, entre outros. Essas informações serão analisadas posteriormente.

O banco de dados *test* é um banco de dados vazio que, como o próprio nome sugere, serve para fazer testes.

Tipos de dados

Use a seguinte legenda para interpretar os tipos de dados números:

M – Indica o tamanho máximo na tela.
D – Indica a precisão (decimal).
[ ] - Valores entre colchetes são opcionais.

Sendo assim, caso queira formatar um número *123.2* com tamanho total (M) = 10 e decimal (D) = 3 o número ficará assim: *123.200* ou caso ZEROFILL tenha sido informado, *000123.200*. O parâmetro *Unsigned* informa se o campo poderá ter sinal ou não.

TINYINT[(M)] [UNSIGNED] [ZEROFILL]
É um inteiro muito pequeno, variando com sinal de -128 á 127 ou sem sinal de 0 á 255.

SMALLINT[(M)] [UNSIGNED] [ZEROFILL]
É um inteiro pequeno, variando com sinal de -32.768 á 32.767 ou sem sinal de 0 á 65.535.

MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]
É um inteiro de tamanho médio, variando com sinal de -8.388.608 á 8.388.607 ou sem sinal de 0 á 16.777.215.

INT[(M)] [UNSIGNED] [ZEROFILL], INTEGER[(M)] [UNSIGNED] [ZEROFILL]
É um inteiro de tamanho normal, variando com sinal de -2.147.483.648 á 2.147.483.647 ou sem sinal de 0 á 4.294.967.295 .

BIGINT[(M)] [UNSIGNED] [ZEROFILL]
É um inteiro grande, variando com sinal de -9.223.372.036.854.775.808 á 9.223.372.036 .854.775.807 ou sem sinal de 0 á 18.446.74 4.073.709.551.615.

FLOAT[(M, D)] [ZEROFILL]
É um número em ponto-flutuante sinalizado, de precisão simples. Os valores aceitáveis são -3.402823466E+38 à -1.175494351E-38, 0, 1.175494351E-38 á 3.402823466E+38

DOUBLE[(M, D)] [ZEROFILL], DOUBLE PRECISION[(M, D)] [ZEROFILL] REAL[(M, D)] [ZEROFILL]
É um número em ponto-flutuante sinalizado, de precisão dupla. Os valores aceitáveis são -1.7976931348623157E+308 à -2.2250738585072014E-308, 0, 2.2250738585072014E-308 á 1.7976931348623157E+308

DECIMAL[(M [, D])] [ZEROFILL], NUMERIC(M D) [ZEROFILL]
É um número em ponto-flutuante sinalizado e não compactado. É armazenado no banco de dados como texto.

DATE
Uma data, que pode estar entre '1001-01-01' á '9999-12-31'. O MySQL apresenta a data no formato YYYY-MM-DD, onde YYYY é o ano com 4 dígitos, MM o mês, e DD o dia.

DATETIME
É a combinação de data e hora, que pode estar entre '1001-01-01 00:00:00' á '9999-12-31 23:59:59'. O MySQL apresenta o tipo no formato YYYY-MM-DD HH:MM:SS, onde HH é a hora (no formato 24 horas, de 0 a 23), MM os minutos (de 0-59), e SS os segundos (de 0-59).

TIMESTAMP[(M)]
É a combinação de data e hora como um DATETIME, mas sua faixa de valores é reduzida, indo de '1970-01-01 00:00:00' até o ano de 2037. O MySQL apresenta o tipo no formato YYYYDDMMHHMMSS, YYDDMMHHMMSS, YYYYMMDD ou YYMMDD, dependendo de M, que pode ser 14, 12, 8, 6 ou não especificado, sendo que nesse último caso, o valor assumido é 14. Não é necessário fornecer um valor para ele, pode-se atribuir a data e hora corrente simplesmente fornecendo um valor NULL para o campo.

TIME
Uma hora. Sua faixa de valores está entre '-838:59:59' á '838:59:59'. O MySQL apresenta a hora no formato HH:MM:SS.

YEAR[(2 | 4)]
Um ano com 2 ou 4 dígitos. Os valores possíveis estão entre 1.901 á 2.155 em 4 dígitos e 1.970 á 2.069 em 2 dígitos (70-69).

CHAR(M)
Texto de tamanho fixo. Se o valor de M não for atingido, a diferença é preenchida com espaços em branco. Quando um valor é lido, os espaços são omitidos.

BIT, BOOL, CHAR
São equivalentes à CHAR(1).

VARCHAR(M)
Texto de tamanho variável. Os espaços são removidos no armazenamento. M pode variar de 1 a 255 bytes.

TINYBLOB, TINYTEXT
Usados para armazenar dados binários / texto respectivamente. O tamanho máximo é de 255 bytes.

BLOB
TEXT
Tamanho máximo é de 65.535 bytes.

MEDIUMBLOB, MEDIUMTEXT
Tamanho máximo é de 16.777.215 bytes.

LONGBLOB, LONGTEXT
Tamanho máximo é de 4.294.967.295 bytes.

ENUM('valor 1', 'valor 2', ...)
Assume o valor de um string determinado entre uma lista de valores pré-definidos durante a criação da tabela, podendo também ser um string em branco ('') ou NULL. O número máximo de elementos é 65.535.

SET('valor 1', 'valor 2', ...)
É um string vazio, ou composto por um ou mais valores/strings pré-determinados. O número máximo de valores possíveis é 64.

Vale lembrar que alguns tipos que armazenam informações de data e hora têm um comportamento diferente nas versões do MySQL anteriores e posteriores à versão 4.1.

Criando um Banco de Dados

Nesse exemplo, criaremos um banco de dados chamado '*Cadastro*' com uma tabela chamada Clientes. A sintaxe para a criação do banco de dados é a seguinte:

CREATE DATABASE [IF NOT EXISTS] NomeDoBanco

Para criarmos o banco de dados digite o seguinte comando:

```
mysql> create database Cadastro;
Query OK, 1 row affected (0.00 sec)
```

A linha *'Query OK, 1 row affected (0.00 sec)'* indica que o banco de dados foi criando com êxito. Agora teremos que selecionar esse BD para poder-mos utilizá-lo. Para selecioná-lo, digite '*use Cadastro;*', como abaixo:

```
mysql> use Cadastro;
Database changed
mysql>
```

Lembre-se que, no Linux, há diferenciação entre letras maiúsculas e minúsculas.

Criando Tabelas

Agora iremos criar a tabela 'Clientes'. A sintaxe para a criação de tabelas é a seguinte:

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] NomeDaTabela (Definicao, ...)

Definição:

```
NomeDoCampo Tipo  [NOT NULL | NULL] [DEFAULT ValorDefault] [AUTO_INCREMENT]  [PRIMARY KEY] ou
PRIMARY KEY (IndiceDeOutraColuna) ou
KEY [NomeDoIndice] (IndiceDeOutraColuna) ou
INDEX [NomeDoIndice] (IndiceDeOutraColuna) ou
UNIQUE [INDEX ] [NomeDoIndice] (IndiceDeOutraColuna) ou
FULLTEXT [INDEX ] [NomeDoIndice] (IndiceDeOutraColuna)
```

Sendo assim, vamos criar a tabela conforme a estrutura descrita na **listagem 1**.

A sintaxe para a criação da tabela é a seguinte:

```
mysql> create table Clientes (Codigo MEDIUMINT(5) AUTO_INCREMENT PRIMARY KEY, Nome VARCHAR(40), Endereco VARCHAR(50), Bairro VARCHAR(30), Cep VARCHAR(9), Cidade VARCHAR(40), Uf VARCHAR(2), DataCadastro DATE, LimiteCredito FLOAT(10,2) UNSIGNED);
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Para confirmar se a tabela foi criada corretamente, digite o comando '*describe Clientes;*'. O resultado deverá ser semelhante ao da **figura 1**.

```
[Clientes]

Codigo          Inteiro     M=5   sem sinal
Nome            Texto       40
Endereco        Texto       50
Bairro          Texto       30
Cep             Texto       9
Cidade          Texto       40
Uf              Texto       2
DataCadastro    Data
LimiteCredito   Real        M=10 D=2 sem sinal
```

Listagem 1. Estrutura da tabela Clientes

```
mysql> describe Clientes;
+---------------+----------------------+------+-----+---------+----------------+
| Field         | Type                 | Null | Key | Default | Extra          |
+---------------+----------------------+------+-----+---------+----------------+
| Codigo        | mediumint(5)         |      | PRI | NULL    | auto_increment |
| Nome          | varchar(40)          | YES  |     | NULL    |                |
| Endereco      | varchar(50)          | YES  |     | NULL    |                |
| Bairro        | varchar(30)          | YES  |     | NULL    |                |
| Cep           | varchar(9)           | YES  |     | NULL    |                |
| Cidade        | varchar(40)          | YES  |     | NULL    |                |
| Uf            | char(2)              | YES  |     | NULL    |                |
| DataCadastro  | date                 | YES  |     | NULL    |                |
| LimiteCredito | float(10,2) unsigned | YES  |     | NULL    |                |
+---------------+----------------------+------+-----+---------+----------------+
9 rows in set (0.00 sec)
```

Figura 1. Describe aplicado sobre a tabela Clientes

Inserindo dados

A sintaxe para inserção de dados é a seguinte:

```
INSERT INTO NomeDaTabela SET Campo1 =
Valor1, Campo2 = Valor2, ...
```

Sendo assim, vamos inserir três clientes fictícios na tabela Clientes:

```
Nome            Cristina Malhas Ltda.
Endereco        R. Getúlio, 1020
Bairro          Vila Nova
Cep             12345-222
Cidade          São Paulo
Uf              SP
DataCadastro    22/08/2002
LimiteCredito   R$ 1500.00

Nome            Frangos Avenida Ltda.
Endereco        R. Santos, 315
Bairro          Bom Retiro
Cep             54321-222
Cidade          Joinville
Uf              SC
DataCadastro    14/07/1996
LimiteCredito   R$ 3000.00

Nome            Supermercados Alvorada
Ltda.
Endereco        R. Gomes Lima, 1220
Bairro          Gonzaga
Cep             04435-111
Cidade          Santos
Uf              SP
DataCadastro    02/04/2000
LimiteCredito   R$ 2000.00
```

As instruções de inserção desses três clientes são:

```
mysql> insert into Clientes set
Nome='Cristina Malhas Ltda.',
Endereco='R. Getulio, 1020',
Bairro='Vila Nova', Cep='12345-222',
Cidade='Sao Paulo', Uf='SP', DataCa-
dastro=20020822, LimiteCredito=1500;
Query OK, 1 row affected (0.00 sec)

mysql> insert into Clientes set
Nome='Frangos Avenida Ltda.',
Endereco='R. Santos, 315',
Bairro='Bom Retiro', Cep='54321-222',
Cidade='Joinville', Uf='SC', DataCa-
dastro=19960714, LimiteCredito=3000;
Query OK, 1 row affected (0.00 sec)

mysql> insert into Clientes set
Nome='Supermercados Alvorada Ltda.',
Endereco='R. Gomes Lima, 1220',
Bairro='Gonzaga', Cep='04435-111',
Cidade='Santos', Uf='SP', DataCadas-
tro=20000402, LimiteCredito=2000;
Query OK, 1 row affected (0.00 sec)
```

Verifique que ao inserirmos dados, o formato da data foi invertido, isto é, no momento de inserção, devemos usar o formato YYYYMMDD. Não informamos valor algum para o campo *Código*, pois com a opção **auto_increment** é atribuído um valor seqüencial ao campo, iniciando com zero.

Selecionando Registros

Para selecionarmos registros nas tabelas, utilizamos a função select, cuja sintaxe é a seguinte:

```
SELECT expressão [FROM NomeDaTabela
       [WHERE condição]
       [GROUP BY campo [ASC | DESC]]
       [HAVING condição]
       [ORDER BY campo [ASC | DESC]]
       [LIMIT registros]
```

Expressão pode ser um (*) para recuperar todos os campos, ou (campo1, campo2, ...) caso queira selecionar campos específicos,

ou mesmo uma expressão, como a seguinte:

```
mysql> select 1 + 1;
+-------+
| 1 + 1 |
+-------+
|     2 |
+-------+
1 row in set (0.07 sec)
```

Podem-se fazer expressões matemáticas, lógicas, entre outros. No exemplo mostrado da **figura 2**, estamos selecionando os campos Codigo, Nome, DataCadastro e Limite de Crédito.

Repare como o MySQL apresenta a data, usando o formato YYYY-MM-DD.

Excluindo Registros

A sintaxe para exclusão de registros é a seguinte:

```
DELETE [LOW PRIORITY | QUICK] FROM
NomeDaTabela
  [WHERE Codicao]
  [ORDER BY (Campo1, Campo2, ...)
[ASC | DESC]]
  [LIMIT NoRegistros]
```

A condição LOW PRIORITY ou QUICK refere-se a prioridade deve ser feita a exclusão. Podemos excluir registros limitando-se o número total de registros com a cláusula **LIMIT**.

Para eliminar o registro número 3, utilizaremos o seguinte comando sql:

```
mysql> delete from Clientes where Co-
digo=3;
Query OK, 1 row affected (0.00 sec)
```

Poderíamos eliminar todos os registros da tabela usando o seguinte comando:

```
mysql> delete from Clientes;
Query OK, 0 rows affected (0.43 sec)
```

Neste caso, o campo Código que é um inteiro auto-incremental, retornará o valor inicial para zero. Se for excluído registros com uma condição **where**, o valor do campo código continuará sendo o valor da última inserção mais um.

Alterando Registros

A sintaxe para a alteração de dados em registros é a seguinte:

```
UPDATE [LOW PRIORITY] [IGNORE]
NomeDaTabela
SET    Campo1 = Valor1,
       Campo2 = Valor2, ...
WHERE Condição
LIMIT NoRegistros
```

LOW PRIORITY faz com que o comando tenha uma prioridade baixa de execução. **IGNORE** é utilizado para que o comando continue executando caso tenha encontrado erros ao atualizar um ou mais registros.

Para demonstrar, vamos alterar a cidade dos registros cujo código são 2 e 3 para Curitiba:

```
mysql> update Clientes set
Cidade="Curitiba" where Codigo >= 2;
Query OK, 2 rows affected (0.00 sec)

Rows matched: 2  Changed: 2  Warnin-
gs: 0
```

Segurança

Este tópico merece atenção especial, pois define como será a forma de acesso ao banco e quais são os bancos que poderão ser acessados. Essas informações são definidas no banco de dados mysql, já existente. Lembre-se: o MySQL já vem com dois bancos de dados padrões - mysql e test.

Digite o comando '*use mysql;*' para conectar-se ao banco de dados **mysql**, e execute o comando '*show tables;*' para visualizar as tabelas, conforme abaixo:

```
mysql> use mysql;
Reading table information for comple-
tion of table and column names
You can turn off this feature to get
```

```
mysql> select Codigo, Nome, DataCadastro, LimiteCredito from Clientes;
+--------+------------------------------+--------------+---------------+
| Codigo | Nome                         | DataCadastro | LimiteCredito |
+--------+------------------------------+--------------+---------------+
|      1 | Cristina Malhas Ltda.        | 2002-08-22   |       1500.00 |
|      2 | Frangos Avenida Ltda.        | 1996-07-14   |       3000.00 |
|      3 | Supermercados Alvorada Ltda. | 2000-04-02   |       2000.00 |
+--------+------------------------------+--------------+---------------+
3 rows in set (0.00 sec)
```

Figura 2. Resultado do select aplicado na tabela Clientes

```
a quicker startup with -A

Database changed
mysql> show tables;
+-----------------+
| Tables_in_mysql |
+-----------------+
| columns_priv    |
| db              |
| func            |
| host            |
| tables_priv     |
| user            |
+-----------------+
6 rows in set (0.00 sec)
```

Como podemos ver, existem 6 tabelas definidas no banco de dados **mysql**. Estudaremos três tabelas específicas: **db, host** e **user**.

Tabela "user"

Use a função '*describe user;*' que mostrará a estrutura dessa tabela, conforme **figura 3**.

O *campo user* é utilizado para definir qual usuário terá acesso. O *campo host* serve para definir de onde este usuário poderá conectar-se (endereço IP). Por exemplo, podemos definir o campo Host como '192.168.0.1', sendo assim o usuário poderá conectar-se somente a partir da máquina cujo endereço é 192.168.0.1. Podemos definir que o usuário poderá conectar-se a partir de qualquer máquina, para isso definimos o campo Host com o valor **'%'**. O *campo password* é utilizado para armazenar a senha para conexão. Esta senha pode ser criptografada utilizando a função PASSWORD() existente no mysql. Os campos *Select_priv, Insert_priv, Update_priv e Delete_priv* servem para definir se os usuários poderão fazer seleções de registros com o comando *select*, inserções com *insert*, alterações com *update,* ou exclusões com *delete*. Os campos *Create_priv* e *Drop_priv* informam se o usuário poderá criar ou apagar tabelas. O campo *Alter_priv* informa se o usuário poderá alterar a estrutura da tabela.

```
mysql> describe user;
+-----------------+---------------+------+-----+---------+-------+
| Field           | Type          | Null | Key | Default | Extra |
+-----------------+---------------+------+-----+---------+-------+
| Host            | char(60)      |      | PRI |         |       |
| User            | char(16)      |      | PRI |         |       |
| Password        | char(16)      |      |     |         |       |
| Select_priv     | enum('N','Y') |      |     | N       |       |
| Insert_priv     | enum('N','Y') |      |     | N       |       |
| Update_priv     | enum('N','Y') |      |     | N       |       |
| Delete_priv     | enum('N','Y') |      |     | N       |       |
| Create_priv     | enum('N','Y') |      |     | N       |       |
| Drop_priv       | enum('N','Y') |      |     | N       |       |
| Reload_priv     | enum('N','Y') |      |     | N       |       |
| Shutdown_priv   | enum('N','Y') |      |     | N       |       |
| Process_priv    | enum('N','Y') |      |     | N       |       |
| File_priv       | enum('N','Y') |      |     | N       |       |
| Grant_priv      | enum('N','Y') |      |     | N       |       |
| References_priv | enum('N','Y') |      |     | N       |       |
| Index_priv      | enum('N','Y') |      |     | N       |       |
| Alter_priv      | enum('N','Y') |      |     | N       |       |
+-----------------+---------------+------+-----+---------+-------+
17 rows in set (0.00 sec)
```

Figura 3. Estrutura da tabela user

Tabela "db"

Use a função 'describe db;' que mostrará a estrutura da tabela db, conforme **figura 4**.

A diferença entre a *tabela db* e a *user* é que em *db* o campo *Password* foi removido, e adicionado o campo *Db*. Todas as tabelas são utilizadas em conjunto, Mostraremos como integrar as tabelas em seguida. O *campo Db* é utilizado para armazenar o nome do banco de dados que terá acesso, no nosso caso o banco '*Cadastro*'.

```
mysql> describe db;
+-----------------+---------------+------+-----+---------+-------+
| Field           | Type          | Null | Key | Default | Extra |
+-----------------+---------------+------+-----+---------+-------+
| Host            | char(60)      |      | PRI |         |       |
| Db              | char(32)      |      | PRI |         |       |
| User            | char(16)      |      | PRI |         |       |
| Select_priv     | enum('N','Y') |      |     | N       |       |
| Insert_priv     | enum('N','Y') |      |     | N       |       |
| Update_priv     | enum('N','Y') |      |     | N       |       |
| Delete_priv     | enum('N','Y') |      |     | N       |       |
| Create_priv     | enum('N','Y') |      |     | N       |       |
| Drop_priv       | enum('N','Y') |      |     | N       |       |
| Grant_priv      | enum('N','Y') |      |     | N       |       |
| References_priv | enum('N','Y') |      |     | N       |       |
| Index_priv      | enum('N','Y') |      |     | N       |       |
| Alter_priv      | enum('N','Y') |      |     | N       |       |
+-----------------+---------------+------+-----+---------+-------+
13 rows in set (0.00 sec)
```

Figura 4. Estrutura da tabela db

exclusões, mas **não** poderão criar, alterar ou excluir tabelas. Para isso definiremos o usuário *Master*. As definições são feitas nas tabelas *user* e *db*. Verifique que na tabela *user* pode-se definir um *Host*, *User* e *Password,* mas não o banco de dados. Isto é, o usuário cadastrado na tabela *user* tem acesso a todos os bancos de dados. Para resolver isto, vamos cadastrar os usuários, mas não daremos nenhum acesso a eles nessa tabela. As senhas para os usuários serão:

Usuário: Joao	Senha: john
Usuário: Maria	Senha: mary
Usuário: Master	Senha: total

A seguir temos os comandos insert para definição dos usuários e senhas:

Tabela "host"

Use a função '*describe db;*' que mostrará a estrutura da tabela db, conforme a **figura 5**.

Observa que na tabela host, não existe o campo *user.*

Definindo a política de segurança

Para definirmos a política de segurança, temos que definir usuários para acesso ao sistema. Definiremos três usuários: *Joao, Maria* e *Master*. João terá acesso ao sistema a partir de qualquer máquina, e Maria somente a partir da máquina cujo IP é 192.168.0.20. Tanto João como Maria, terão acesso para fazer seleções, inserções, atualizações e

```
mysql> describe host;
+-----------------+---------------+------+-----+---------+-------+
| Field           | Type          | Null | Key | Default | Extra |
+-----------------+---------------+------+-----+---------+-------+
| Host            | char(60)      |      | PRI |         |       |
| Db              | char(32)      |      | PRI |         |       |
| Select_priv     | enum('N','Y') |      |     | N       |       |
| Insert_priv     | enum('N','Y') |      |     | N       |       |
| Update_priv     | enum('N','Y') |      |     | N       |       |
| Delete_priv     | enum('N','Y') |      |     | N       |       |
| Create_priv     | enum('N','Y') |      |     | N       |       |
| Drop_priv       | enum('N','Y') |      |     | N       |       |
| Grant_priv      | enum('N','Y') |      |     | N       |       |
| References_priv | enum('N','Y') |      |     | N       |       |
| Index_priv      | enum('N','Y') |      |     | N       |       |
| Alter_priv      | enum('N','Y') |      |     | N       |       |
+-----------------+---------------+------+-----+---------+-------+
12 rows in set (0.05 sec)
```

Listagem 5. Estrutura da tabela host

```
mysql> insert into user values ('%',
'Joao', Password('john'), 'N', 'N',
'N', 'N', 'N', 'N', 'N', 'N', 'N',
'N', 'N', 'N', 'N', 'N');
Query OK, 1 row affected (0.49 sec)

mysql> insert into user values ('%',
'Maria', Password('mary'), 'N', 'N',
'N', 'N', 'N', 'N', 'N', 'N', 'N',
'N', 'N', 'N', 'N', 'N');
Query OK, 1 row affected (0.00 sec)

mysql> insert into user values ('%',
'Master', Password('total'), 'N',
'N', 'N', 'N', 'N', 'N', 'N', 'N',
'N', 'N', 'N', 'N', 'N', 'N');
Query OK, 1 row affected (0.00 sec)
```

Agora que criamos os usuários, faremos as devidas permissões na tabela *db*. Note que, no exemplo anterior, utilizamos a função ***Password('Senha')*** para cadastrar senhas seguras em nosso banco de dados.

```
mysql> insert into db values ('%',
'Cadastro', 'Joao', 'Y', 'Y', 'Y',
'Y', 'N', 'N', 'N', 'N', 'N', 'N');
Query OK, 1 row affected (0.41 sec)

mysql> insert into db values
('192.168.0.20', 'Cadastro', 'Maria',
'Y', 'Y', 'Y', 'Y', 'N', 'N', 'N',
'N', 'N', 'N');
Query OK, 1 row affected (0.00 sec)

mysql> insert into db values ('%',
'Cadastro', 'Master', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y');
Query OK, 1 row affected (0.01 sec)

mysql>
```

Verifique que *Maria* poderá se conectar-se somente do host 192.168.0.20, e que o usuário *Master* poderá fazer quaisquer alterações, e que todos só poderão se conectar ao banco de dados *Cadastro*.

Digite '*flush privileges;*' para **ativar** as permissões conforme o exemplo abaixo:

```
mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Digite o comando 'quit' para sair do mysql client. Agora vamos conectar como os usuários cadastrados.

Para conectar como um usuário específico, utilize a sintaxe abaixo:

```
mysql -u usuario -p
```

A opção **-u** solicita que seja informado um nome de usuário. A opção **-p** solicitará a senha assim que o comando for executado, conforme visto abaixo:

```
shell> mysql -u Joao -p
Enter password:
```

Pode-se também utilizar a opção **-h host** caso o banco de dados não esteja na máquina local, conforme a seguir:

```
shell> mysql -u 192.168.0.1 -u Joao
-p
Enter password:
```

Importando dados a partir de um arquivo

A importação de dados a partir de um arquivo texto para uma tabela é muito útil, especialmente quando se está migrando uma aplicação já existente, para um banco de dados MySQL. A sintaxe é a seguinte:

```
LOAD DATA INFILE 'dados.txt' INTO
TABLE NomDaTabela
      FIELDS TERMINATED BY ';'
      LINES TERMINATED BY '\n'
```

No exemplo acima, 'dados.txt' representa o arquivo texto do qual será extraído os dados. '

NomeDaTabela é a tabela que conterá os dados extraídos. O ';' (ponto e vírgula) representa os delimitadores de campo. O '\n' é o símbolo de new line que quer dizer nova linha, isto é, cada linha ou cada registro é terminado por '\n'.

Lembre-se que os campos devem seguir a mesma seqüência e coincidir na quantidade. Sendo assim, se tivermos uma tabela com 6 campos, o arquivo texto deverá ter o formato:

```
CAMPO1 ; CAMPO2 ; CAMPO3 ; CAMPO4 ;
CAMPO5 ; CAMPO6 ;
CAMPO1 ; CAMPO2 ; CAMPO3 ; CAMPO4 ;
CAMPO5 ; CAMPO6 ;
CAMPO1 ; CAMPO2 ; CAMPO3 ; CAMPO4 ;
CAMPO5 ; CAMPO6 ;
```

Exportando dados a partir do MySQL

Podemos extrair os dados de uma tabela para um arquivo texto, conforme a sintaxe á seguir:

```
SELECT * INTO OUTFILE 'dados.txt'
       FIELDS TERMINATED BY ';'
       FROM Tabela
```

A exportação utiliza uma consulta *select*, conforme já foi visto, mais a expressão **INTO OUTFILE** 'dados.txt', onde dados.txt é o nome que daremos ao arquivo que será gerado pela consulta. **FIELDS TERMINATED BY** ';' (ponto e vírgula) é a terminação dos campos, e Tabela é a relação de onde serão extraídos os dados. Pode-se ampliar o filtro de dados utilizando uma cláusula *where,* como em uma consulta select normal.

Conclusão

Vimos nesse artigo as informações básicas para qualquer um se iniciar no uso do Banco de Dados MySQL. Não deixe de consultar a documentação oficial do banco e os diversos sites existentes na internet para se aprofundar nos inúmeros recursos desse banco.

Avalie esse artigo		
ÓTIMO	BOM	RUIM

Autor:

Henderson Macedo (henderson@draft-automacao.com.br) – Analista de Sistemas - HMO Sistemas

Mini-curriculo

Bacharel em Sistemas de Informação pela Universidade da Região de Joinville (Joinville / SC). Desenvolveu um sistema de gestão totalmente em Java (Swing) e banco de dados MySQL (S.I.G.E. Network - Sistema Integrado de Gerenciamento Empresarial). Já desenvolveu projetos em Visual Basic, mas atualmente trabalha em tempo integral com Java e Linux.

Acessando bases de dados PostgreSQL no PHP com ADOdb

Marcelo Santos Araújo

ADOdb é uma biblioteca que abstrai o acesso aos bancos de dados, ou seja, ela simplifica inúmeras tarefas relacionadas ao PHP e as bases de dados. A ADOdb API é baseada na Microsoft ADO, uma biblioteca de acesso muito utilizada em Visual Basic e outros produtos da Microsoft. Apesar da existência da PEAR DB (encontrada em http://pear.php.net), a ADOdb apresenta recursos específicos, capazes de amenizar tarefas complexas em bases de dados, como por exemplo, paginação e criação de combobox(es).

A PEAR DB apresenta uma estrutura orientada a objetos mais rigorosa que a ADOdb, bem como maior flexibilidade com outros pacotes da PEAR. Entretanto, quando tratamos de recursos complexos, a melhor escolha acaba sendo a ADOdb.

Instalando a ADOdb

Faça o download do pacote ADOdb no seguinte endereço: http://adodb.sourceforge.net/#download . Os bancos de dados suportados pela ADOdb são: Access, ADO, DB2, FrontBase, Informix, InterBase, Microsoft SQL Server, MySQL, Oracle, ODBC, PostgreSQL, SAPDB, SQLAnywhere, SQLite, Sybase e Visual FoxPro.

Conexão e consultas simples

Esta seção descreve alguns exemplos simples de conexão com o **PostgreSQL** usando a biblioteca ADOdb. A **tabela 1** apresenta os drivers para os respectivos bancos de dados suportados.

Drivers	Banco de Dados
Ado	Database genérica ADO
Access	Microsoft Access
postgres	PostgreSQL 7 (mesmo que postgres7)
postgres7	PostgreSQL 7
db2	DB2
postgres64	PostgreSQL 6.4 ou inferior
ibase	InterBase 6 ou inferior
firebird	InterBase (Versão Firebird)
odbc	DSN ODBC
informix72	Informix 7.2 ou anterior
mysql	MySQL (sem suporte a transações)
mssql	Microsoft SQL Server 7 ou superior

Tabela1. Drivers e bancos de dados suportados pela ADOdb.

O código do **exemplo 1** faz uma conexão com o banco de dados, roda um *select* e retorna o resultado na forma de HTML simples.

```
<?php
// Incluir a biblioteca
require_once('adodb/adodb.inc.php');
// Conexão ao PostgreSQL
$c = &ADONewConnection('postgres');
//PConnect => conexão persistente
$c->PConnect('host=host port=porta
dbname=database user=usuario
password=senha');
$consulta = $c->Execute("SELECT * FROM
agenda");
// Exibição dos dados
while(!$consulta->EOF)
{
   echo "Nome: ".$consulta->fields[0]."  -
".$consulta->fields[1]."<br>";
   $consulta->MoveNext();
}

 // número de registros da tabela AGENDA
$total_registro = $consulta->Re-
cordCount();
 echo "Total de registro(s): ".$total_re-
gistro;
?>
```

Exemplo 1. Conexão e exibição de dados.

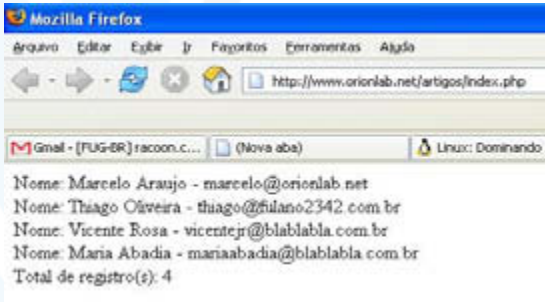

Figura 1. Resultado da execução do exemplo 1.

No **exemplo 2** temos a apresentação do resultado de forma mais elaborada, através do uso de tabelas HTML gerada pela função ***rs2html***. O resultado apresentado no browser pode ser visto na **figura 2**.

```
<?php
// Incluir a biblioteca base
// Incluir a biblioteca de html
require_once("adodb/adodb.inc.php");
require_once("adodb/tohtml.inc.php");

// Conexão ao PostgreSQL
```

PostgreSQL

```
$c = &ADONewConnection("postgres");
//PConnect => conexão persistente

$c->PConnect('host=host port=porta
dbname=database user=usuario
password=senha');
$consulta = $c->Execute("SELECT * FROM
agenda");
rs2html($consulta);
?>
```

Exemplo 2. Exibindo um Record Set em uma tabela HTML.

O **exemplo 3** mostra uma variação do código do exemplo 2, onde personalizamos os nomes apresentados nas colunas da tabela através do uso de um **array** juntamente com a função ***rs2html***.

```
<?php
// Incluir a biblioteca base
// Incluir a biblioteca de html
require_once("adodb/adodb.inc.php");
require_once("adodb/tohtml.inc.php");

// Conexão ao PostgreSQL

$c = &ADONewConnection("postgres");
//PConnect => conexão persistente
$c->PConnect('host=host port=porta dbname=database user=usuario password=senha');
$consulta = $c->Execute("SELECT * FROM agenda");
rs2html($consulta,false,array("Coluna Nome","Coluna Email"));

?>
```

Exemplo 3 – Alterando os nomes das colunas em uma tabela HTML.

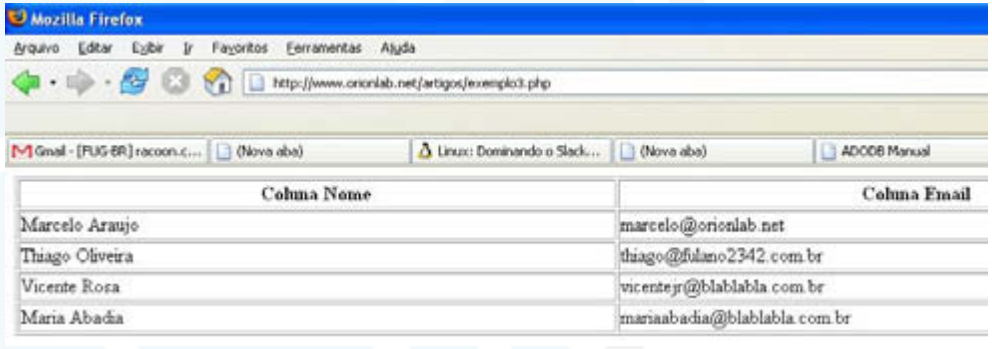

Figura 3. Resultado da execução do código do exemplo 3.

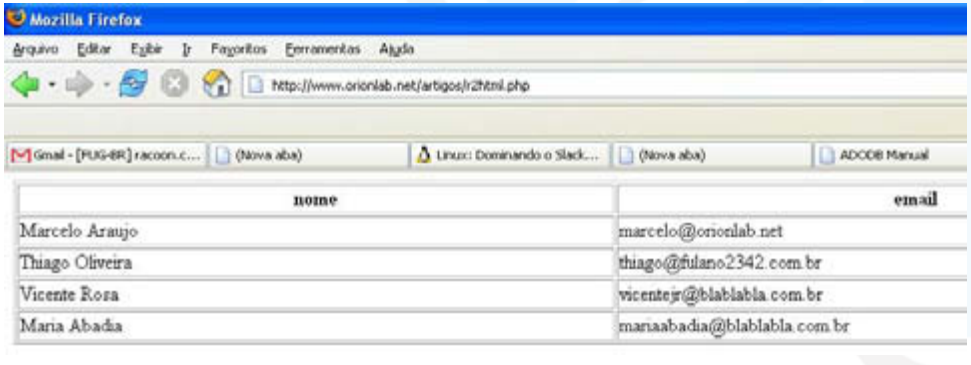

Figura 2. Apresentação dos dados em forma de tabela.

No **exemplo 4** iremos mostrar um combobox no browser, onde os itens visualizados foram recuperados através de um select no BD. O resultado está na **figura 4**.

```
<?php
// Incluir a biblioteca
require_once('adodb/adodb.inc.php');
// Conexão ao PostgreSQL
$c = &ADONewConnection('postgres');
// Conexão Persistente
$c->PConnect('host=host port=porta dbname=database
user=usuario password=senha');
$consulta = $c->Execute("SELECT nome,nome FROM agen-
da");
print $consulta->GetMenu('lista_nome',null,false);

// GetMenu('nome',null,false) - desabilita o option va-
zio do combobox
?>
```

Exemplo 4 – Construção de ComboBox a partir da base de dados.

Agora podemos visualizar nosso combobox:

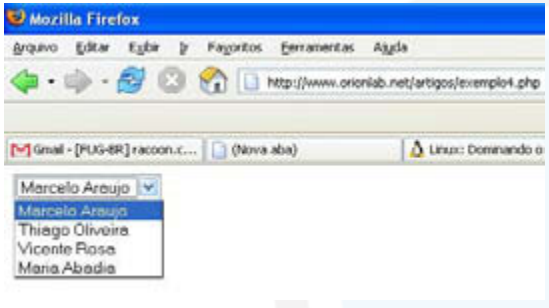

Figura 4. Resultado da execução do código do exemplo 4.

No **exemplo 5**, apresento uma maneira muito fácil de fazer paginação de dados. Isso é útil quando a quantidade de registros retornados pelo select é muito grande e, portanto, fica interessante mostrar o resultado em partes, dando a opção do usuário navegar pelos registros. Verifique a página gerada na **figura 5**.

```
<?php
// Incluir a biblioteca
require_once('adodb/adodb.inc.php');
require_once("adodb/adodb-pager.inc.php");
// Conexão ao PostgreSQL
$c = &ADONewConnection('postgres');
// Conexão Persistente
$c->PConnect('host=host port=porta
dbname=database user=usuario
password=senha');
// Paginação - 2 registros por página
- Render(2)
$paginador = new ADODB_Pager($c,"SELECT
nome,email FROM agenda");
$paginador->Render(2);
?>
```

Exemplo 5. Paginação com ADOdb.

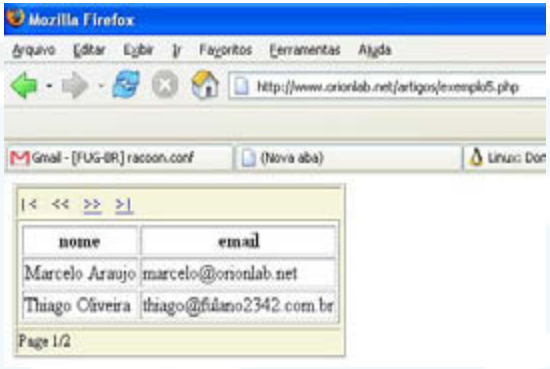

Figura 5. Exemplo de paginação de dados.

Referência das bibliotecas

A seguir listo as bibliotecas utilizadas pelos exemplo, com as respectivas funções:

Biblioteca de paginação: *adodb-pager.inc.php*

Construtor: ADODB_Pager (Paginação)

Método: `Render(4)` – 4 registros por página (por exemplo)

Biblioteca de HTML: *tohtml.inc.php*

`rs2html()` – transforma um Record Set em uma tabela HTML

Biblioteca de conexão: *adodb.inc.php*

`&ADONewConnection($driver)`

(nome alternativo para NewADOConnection)

`NewADOConnection($driver)`

Métodos:

`Connect($host,[$user],[$password],[$database])` *(conexão simples)*

`PConnect($host,[$user],[$password],[$database])` *(conexão persistente)*

Execução direta de comandos SQL

Através do método Execute da ADOdb, podemos enviar comandos SQL diretamente para o servidor de Banco de Dados. Segue alguns exemplos:

INSERT

```
$ok = $db->Execute("INSERT INTO pu-
blic.agenda(nome,email) VALUES('jorge
júnior','jorge@algumhost.com.br')");
```

DELETE

```
$ok = $db->Execute("DELETE FROM pu-
blic.agenda WHERE email='marcelo@
orionlab.net'");
```

UPDATE

```
$ok = $db->Execute("UPDATE public.
agenda SET email='jorge@orionlab.net'
WHERE email = 'jorge@algumhost.com.
br'");
```

PostgreSQL

SELECT

```
$ok = $db->Execute("SELECT * FROM public.agenda");
```

No caso de múltiplas inserções de registros, podemos aumentar muito o desempenho com a ajuda de um *array* contendo os dados a serem inseridos, pois o SQL é preparado uma única vez, e executado diversas vezes, tão quanto for o número de "registros" (elementos do array) a serem inseridos.

Esse método é muito útil para importar dados. O exemplo abaixo faz uso desse recurso:

```
$arr = array(
       array('Vander Pereira',62),
       array('Donivaldo', 23),
       array('Jorge Eurípedes', 21)
       );
$ok = $db->Execute('insert
into tabela(nome,idade) values
(?,?)',$arr);
```

Para maiores detalhes e referências, visite o site com o manual da ADOdb:

http://phplens.com/lens/adodb/docs-adodb.htm

Avalie esse artigo		
ÓTIMO	BOM	RUIM

Autor:
Marcelo Santos Araujo (marcelo@orionlab.net)
Mini-curriculo
Estudante do curso de *Ciência da Computação* da Universidade Federal de Uberlândia (UFU/MG), estagiário do Núcleo de Processamento de Dados (NUPRO) da UFU/MG. Trabalha com PHP, PostgreSQL, MySQL e PEARL.

Review

DBManager

Carlos H. Cantu e
Luiz Paulo de O. Santos

O advento dos sistemas operacionais gráficos, mais especificamente a popularização do Windows, trouxe aos usuários de microcomputadores todas as facilidades que uma GUI (Graphical User Interface) pode oferecer (mouse, botões, janelas, etc). Isso possibilitou o aparecimento de aplicações bem mais amigáveis, em todas as áreas, inclusive na de Bancos de Dados, através de ferramentas gráficas de administração!

Nesse artigo, analisaremos uma dessas ferramentas: o DBManager Pro. Três fatores chamaram minha atenção para esse software:

1. É um software brasileiro;
2. Permite a manutenção de pelo menos 10 bancos de dados diferentes;
3. Possui uma versão FreeWare (com algumas limitações) e uma completa (Enterprise).

Geralmente a maioria das ferramentas para BD que encontramos por aí são estrangeiras, e geralmente suportam um único SGBD. Para os DBAs, que por escolha ou não, trabalham com mais de um SGBD, a possibilidade de gerenciá-los através de uma interface comum é no mínimo bastante atraente.

A seguir veremos alguns dos recursos encontrados no DBManager Pro.

Instalação

Nesse artigo estaremos usando a versão TRIAL do DBManager Enterprise, que funciona por 30 dias, com todos os recursos habilitados.

Para fazer a instalação, acesse o site www.dbtools.com.br e baixe o arquivo correspondente. O processo de instalação é simples. A **figura 1** mostra a tela de seleção de componentes para os diversos SGBDs suportados pelo DBManager. O instalador irá instalar as bibliotecas cliente dos SGBDs que podem ser redistribuídas. Para as outras, você já deve ter o cliente instalado na máquina.

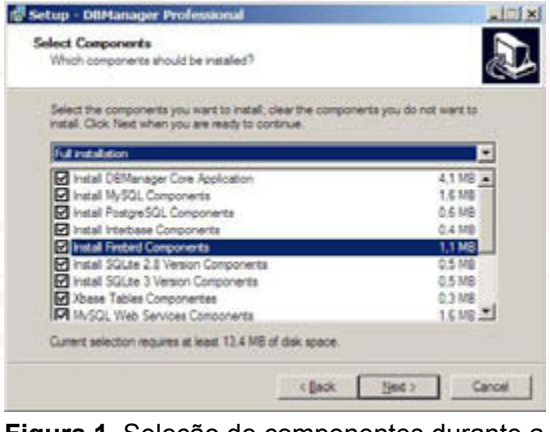

Figura 1. Seleção de componentes durante a instalação.

Registrando um Servidor e um BD

Uma das facilidades que o DBManager oferece é a existência de "wizards" para guiar o usuário em algumas tarefas básicas, como por exemplo o registro de um novo servidor. Na guia *START* da tela principal do programa, clique em "*Add a new Server*" para registrar um novo servidor. No exemplo, estarei registrando um servidor Firebird, conforme a configuração mostrada na **figura 2**. Depois de registrado o servidor, adicionaremos o banco de dados *Employee.fdb* (que acompanha o Firebird) na lista de BDs registrados para esse servidor, configurando-o conforme a **figura 3**.

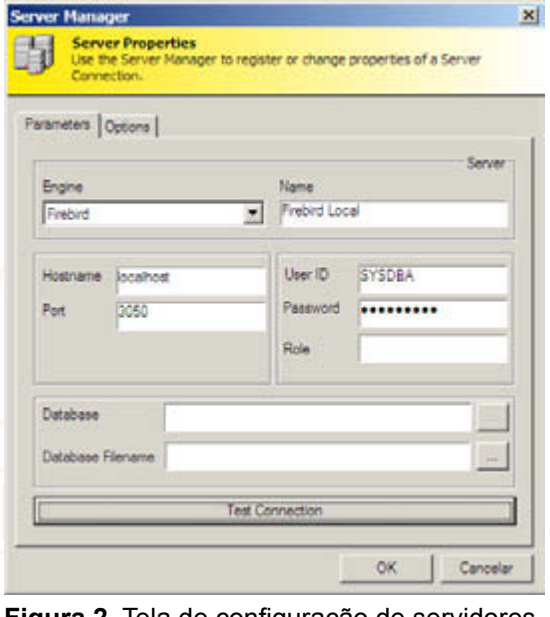

Figura 2. Tela de configuração de servidores

(continua na página seguinte...)

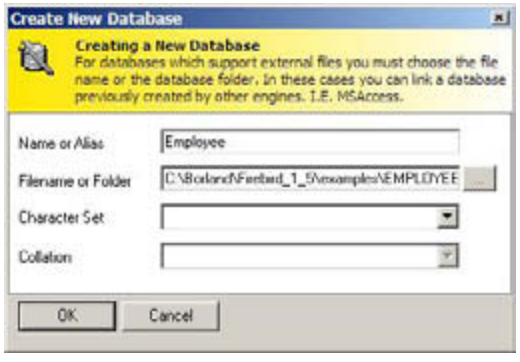

Figura 3. Registro do BD Employee.fdb

Navegando pelo BD

Com o banco de dados já registrado, podemos navegar pelos objetos (tabelas, procedures, views, etc.). Para mostrar algumas das funcionalidades do DBManager, exploraremos a tabela CUSTOMER. A **figura 4** mostra a tela de propriedades da referida tabela. Nela temos as opções de executar algumas tarefas de administração, como editar as propriedades da tabela, visualizar e editar os dados, exportar os dados, imprimir a estrutura, limpar a tabela, removê-la, etc. Através das guias *Columns, Indexes, Checks, Foreign Keys* e *Triggers*, podemos visualizar a estrutura, dependências, regras de integridade, etc.

Clicando no item *Table Properties*, podemos alterar a estrutura da tabela, adicionando *triggers*, regras de integridade, índices, etc. O item *Table Designer* permite adicionar, alterar ou remover campos nessa tabela.

Figura 4. Guia "properties" da tabela CUSTOMER

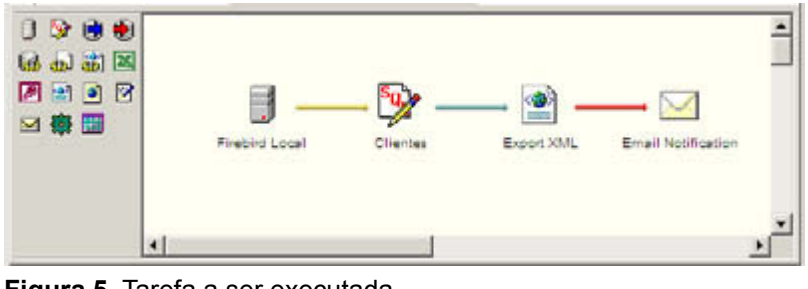

Figura 5. Tarefa a ser executada

Task Builder

Um recurso muito interessante do DBManager é o *Task Builder*. Com ele, você pode configurar tarefas compostas por uma seqüência de ações pré-definidas, como por exemplo, conectar-se a um servidor, recuperar os dados através de uma query, exportar esses dados para um arquivo XML, e enviar esse arquivo por e-mail para uma determinada pessoa. Um exemplo de um diagrama para realizar essa tarefa pode ser visto na **figura 5**. As tarefas já configuradas podem ser executadas através de linha de comando, facilitando o processo de automatização.

Console

Para os que gostam de "botar a mão na massa", o DBManager oferece um console onde comandos SQL podem ser digitados e executados, obtendo-se os resultados imediatamente. A **figura 6** mostra a execução de uma query na tabela *customer* diretamente pelo console.

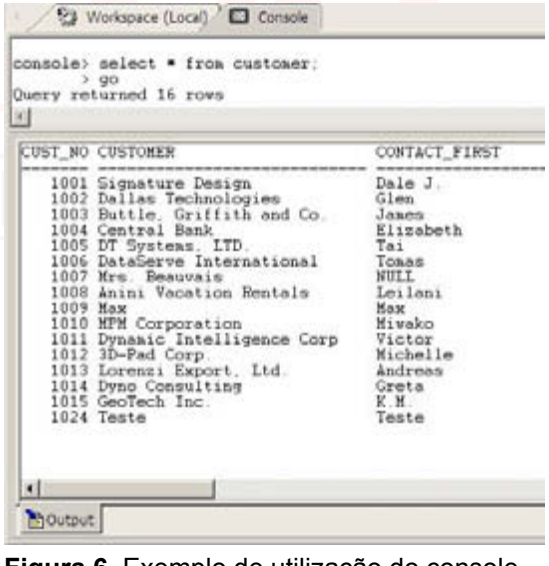

Figura 6. Exemplo de utilização do console

Diagram Designer ou Database Designer

A ferramenta também oferece uma interface para modelagem dos bancos chamada *Diagram Designer (ou Database Designer)*. No caso de bancos já existentes, pode-se facilmente adicionar as tabelas ao diagrama através do menu popup, usando a opção "Add table to diagram".

Novas tabelas podem ser criadas através da opção "*New Table*", e para criar relacionamentos entre elas, basta arrastar e soltar os campos relacionados, criando a ligação entre eles. A **figura 7** mostra um diagrama baseado no banco *employee.fdb*.

Monitor de Servidor e Banco de Dados

Com o Monitor, podemos monitorar as condições de uso do servidor e dos bancos de dados, como por exemplo:

- Montante de memória
- Conexões ativas
- Alocação de buffers
- Status de memória

A **figura 8** mostra um exemplo desse recurso quando conectado a um servidor Firebird. Vale lembrar que as informações apresentadas, bem como seu detalhamento, variam de acordo com o SGBD utilizado.

Database Wizard

A ferramenta oferece wizards que aumentam a produtividade, facilitando a execução de diversas tarefas, como visto na **figura 9**. A **tabela 1** descreve cada um dos itens disponíveis no wizard.

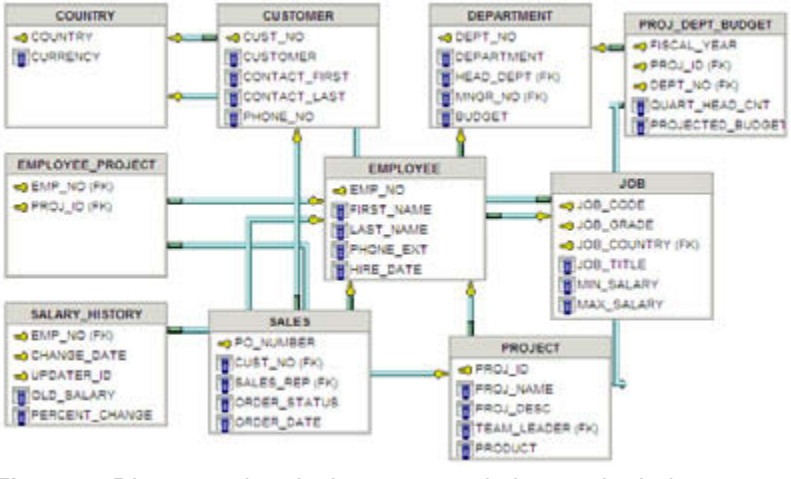

Figura 7. Diagrama de relacionamentos do banco de dados employee.fdb

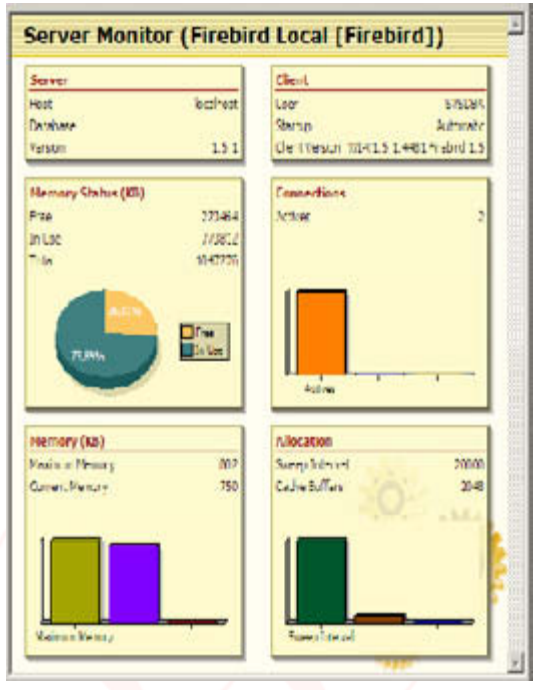

Figura 8. Informações de uma base MS-

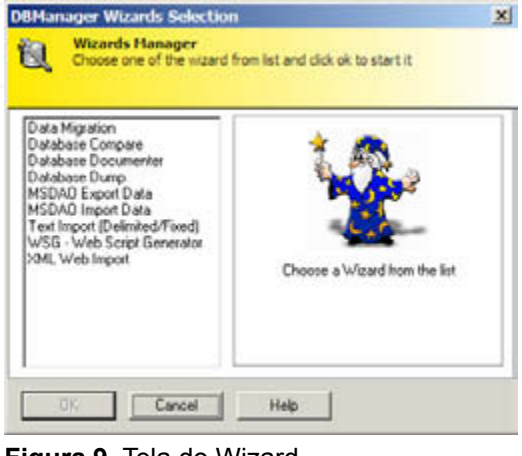

Figura 9. Tela do Wizard

Data Migration	Permite migrar dados e estruturas entre bancos de dados.
Database Compare	Compara e sincroniza dois bancos de dados. O script com as diferenças pode ser executado ou gravado em arquivos SQL/DDL.
Database Documenter	Cria documentação para seus bancos de dados
Database Dump	Permite a criação de scripts SQL com o conteúdo de tabelas de um banco.
MSDAO Export Data	Exporta dados e estruturas para uma base MS-Access
MSDAO Import Data	Importa bases de dados a partir de bases MSAcess, MSExcell, dBase III, IV e V, FoxPro, Paradox e acessos ODBC.
Text Import (Delimited/ Fixed)	Importa os dados de arquivos texto (com tamanho fixo, ou delimitados por vírgula)
WSG – Web Script Generator	Cria scripts PHP, ASP e Perl para conexão ao banco para aplicações baseadas em Web.
XML Web Importer	Importa dados de um Web Script XML ou de um XML gravado em um disco.

Tabela 1. Opções do Database Wizard

Database DUMP

Vejamos como proceder para fazer um DUMP do banco de dados Employee.fdb.

Escolha a opção Database Dump e selecione as tabelas desejadas, conforme a **figura 10**.

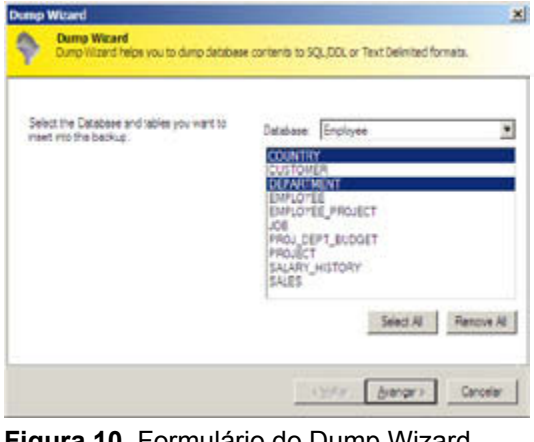

Figura 10. Formulário do Dump Wizard

Após a seleção das tabelas, devemos indicar o que desejamos exportar. As opções podem ser vistas na **figura 11**.

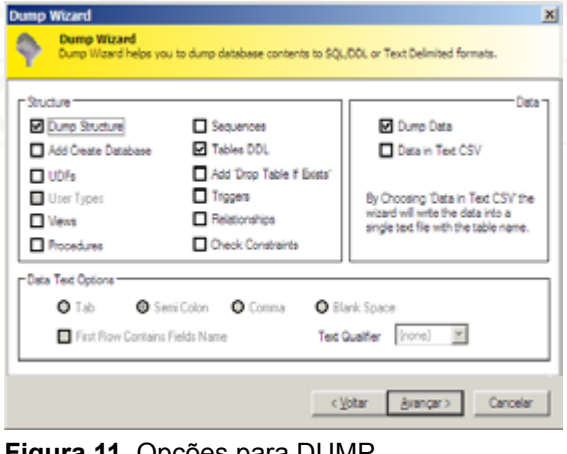

Figura 11. Opções para DUMP

Clicando em *Avançar*, devemos informar o nome do arquivo de script que desejamos criar (exemplo: *c:\dump.txt)* e clicar no botão *SAVE*, informando também um título para o DUMP. Clicando em concluir, o script é gerado. A seguir segue exemplo do arquivo gerado:

```
-- DBTools Firebird - Firebird Database
Dump
-- Dumping Table Structure for COUNTRY

CREATE TABLE "COUNTRY"
(
   "COUNTRY" varchar(15) NOT NULL,
   "CURRENCY" varchar(10) NOT NULL,
   CONSTRAINT RDB$PRIMARY1 PRIMARY KEY (
"COUNTRY" )
);

-- Dumping Data for COUNTRY
-- Dumping Table Structure for DEPARTMENT

CREATE TABLE "DEPARTMENT"
(
   "DEPT_NO" char(3) NOT NULL,
   "DEPARTMENT" varchar(25) NOT NULL,
   "HEAD_DEPT" char(3),
   "MNGR_NO" smallint,
   "BUDGET" decimal(12,2),
   "LOCATION" varchar(15),
   "PHONE_NO" varchar(20) DEFAULT '555-
1234',
   CONSTRAINT RDB$PRIMARY5 PRIMARY KEY (
"DEPT_NO" )
);

CREATE DESC INDEX "BUDGETX" ON "DEPART-
MENT" ( "BUDGET" );
ALTER TABLE "DEPARTMENT" ADD CONSTRAINT
"INTEG_17" FOREIGN KEY ( "HEAD_DEPT" )
REFERENCES "DEPARTMENT" ( "DEPT_NO" );
ALTER TABLE "DEPARTMENT" ADD CONSTRAINT
"INTEG_31" FOREIGN KEY ( "MNGR_NO" ) REF-
ERENCES "EMPLOYEE" ( "EMP_NO" );

-- Dumping Data for DEPARTMENT
-- End of Database Dumping.
```

Web Script Generator

Esse *wizard* é destinado aos desenvolvedores Web que necessitam criar scripts de conexão, autenticação, busca ou atualizações de bases de dados. A tela inicial pode ser vista na **figura 12**. As opções de scripts fornecidas são:

- Busca e exibição de registros (Search and Display Record)
- Encontrar e Editar Registros (Find and Edit Record)
- Serviço Web baseado em XML.

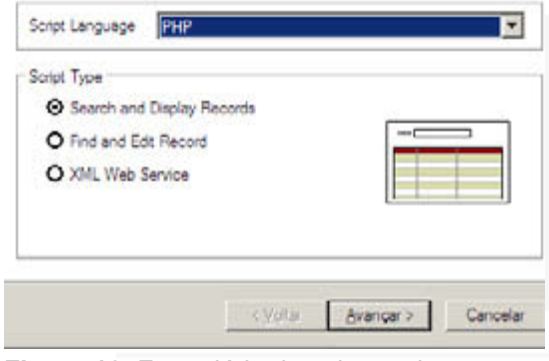

Figura 12. Formulário de seleção de linguagem a gerar o Script

Nesse exemplo, vamos utilizar a primeira opção (*Search and Display Records*). Clique em avançar e selecione o banco e a(s) tabela(s) que deseja incluir no script (ver figura 14). Na próxima tela podemos também indicar quais os tipos de controles web serão usados para cada campo das tabelas selecionadas (ver **figura 13**).

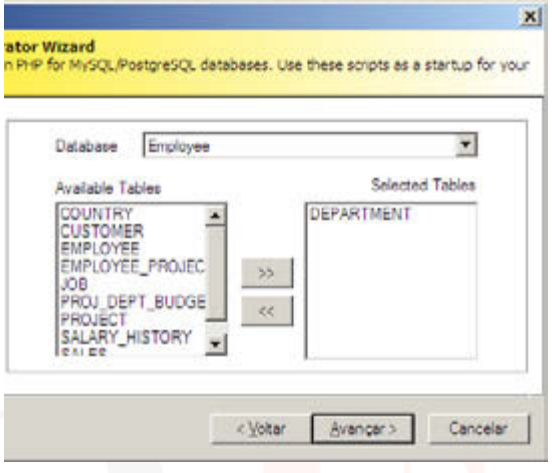

Figura 13. Seleção do BD e tabelas para geração do script

Após a confirmação, teremos o script gerado. Note que você deve alterar o código manualmente e especificar a senha e o usuário de conexão ao SGBD. A **figura 14** mostra o script sendo executado no browser.

DataBase Compare

O *Database Compare* permite comparar dois bancos de dados, obtendo todas as diferenças entre ambos.

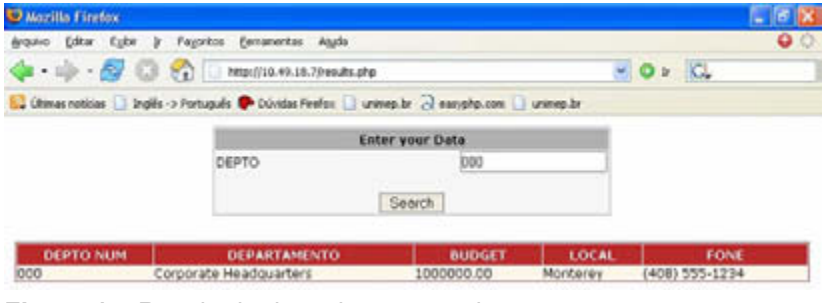

Figura 14. Resultado do script executado

Supondo que tenhamos a base original EMPLOYEE e uma outra base mais atual, chamada COPIA, e desejamos saber quais foram as diferenças implementadas entre as duas bases, basta para isso indicar como SOURCE a base original e como TARGET a base atualizada, conforme a **figura 15**.

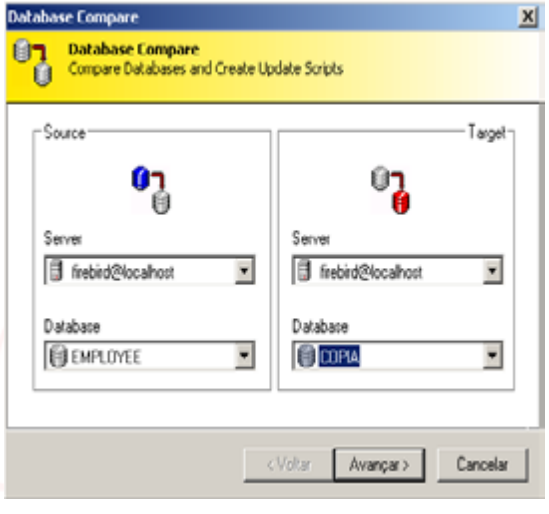

Figura 15. Indicando os bancos que devem ser comparados.

Clicando em *Avançar,* devemos indicar se desejamos comparar todos os objetos, caso contrário devemos informar quais objetos serão comparados. A última pergunta é se desejamos comparar triggers, functions, indexes e foreign keys. Após concluir o processo, teremos uma tela final de resultado semelhante a **figura 16**.

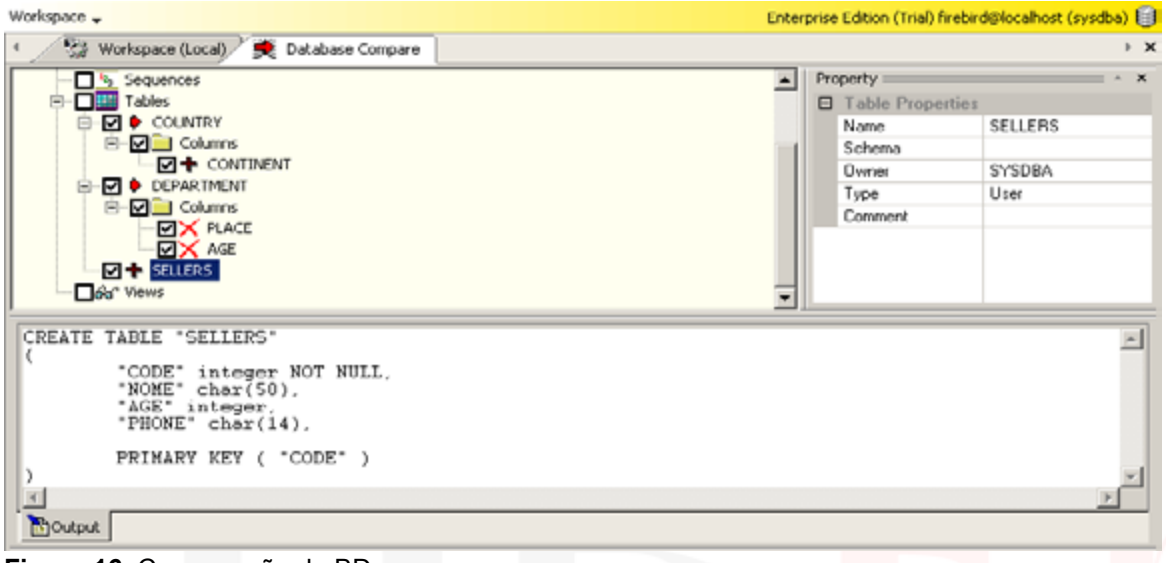

Figura 16. Comparação de BDs

Nela é indicado as diferenças na estrutura dos bancos, e sugerido o código SQL referente a atualização necessária.

. Report Builder

O DBManager possui um gerador de relatórios (Report Builder) que possibilita a confecção de relatórios baseados em queries. Pode-se inclusive criar níveis de agrupamento dos dados, e até mesmo ter a saída do relatório no formato HTML. A **figura 17** mostra o exemplo de um relatório criado no Report Builder.

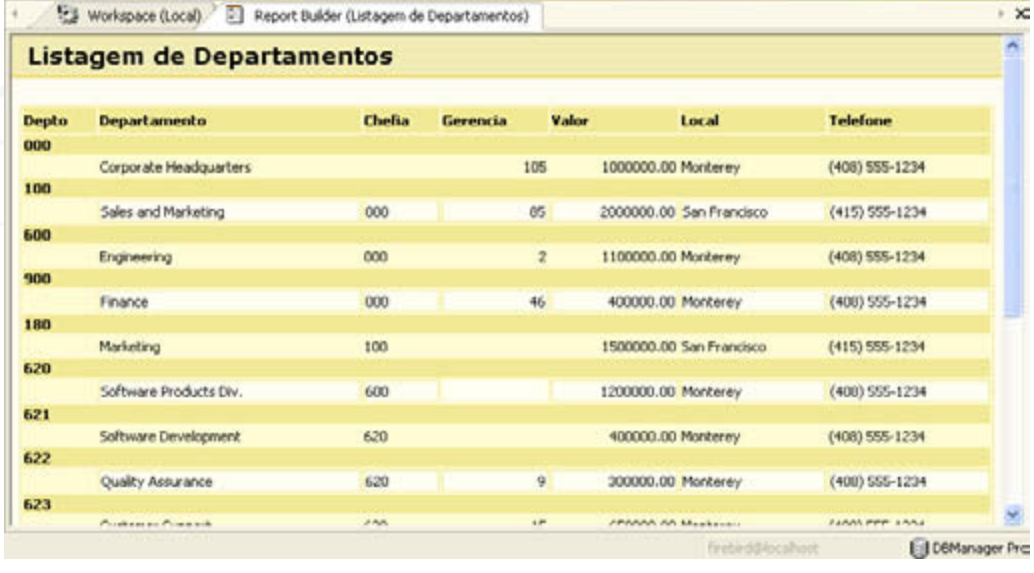

Figura 17. Relatório criado no Report Builder

Form Designer

Através do Form Designer, podemos criar em apenas alguns segundos uma interface básica para consulta, edição e manutenção de dados. Um exemplo de formulário gerado pode ser visto na **figura 17**.

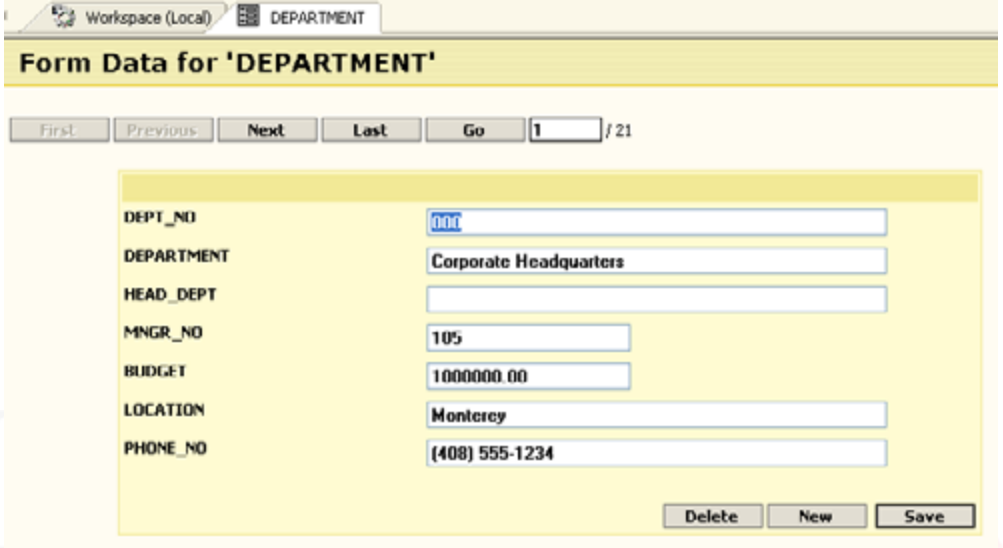

Figura 17. Formulário de edição de dados gerado no Form Designer

Conclusão

A ferramenta oferece recursos muito interessantes, e tem a grande vantagem de possibilitar acesso a diversos SGBDs através da mesma interface. Além disso, é um dos poucos softwares nacionais voltado para a área de administração de BDs.

Alguns detalhes ainda precisam ser melhorados, incluindo a correção de alguns problemas no Query Editor, que nos nossos testes gerou SQL inválido devido ao uso de aspas duplas no nome dos objetos, além de outras pequenas imperfeições, o que deve ocorrer nos próximos releases de manutenção.

Uma interface em português também seria interessante, pois facilitaria ainda mais a vida do pessoal que não é “bom no inglês”.

Se você está procurando por uma ferramenta de manutenção de bancos de dados, não deixe de testar o DBManager, e lembre-se: a versão *Professional* é **gratuita**!

Avalie esse artigo		
ÓTIMO	BOM	RUIM

Calendário de Eventos

Data	Evento	Site
19 à 23 Setembro	XIX sem. Paraense de Inf. e Telecomunicações Belém – PA	http://www.sepai.com.br
23 e 24 de Setembro	2º Encontro de Software de Livre de Ourinhos Ourinhos – SP	http://www.linuxplace.com.br/squsp_place/1124751321/index_html
28/Setembro à 01/Outubro	II Conferência Latino-americana e do Caribe sobre Desenvolvimento e Uso do Software Livre Recife - Olinda - PE	http://twiki.im.ufba.br/bin/view/LacFREE/WebHome
03 à 07 de Outubro	20. Simpósio Brasileiro de Banco de Dados Uberlândia - MG	http://www.sbbd-sbes2005.ufu.br/imagens/ChamadaSBBD.pdf
3,4,5 de Novembro	Conisli – Congresso Internacional de Software Livre São Paulo - SP	http://www.eventosucesusp.org.br/conisli/

*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.*