Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

Manipulação direta de dados

Aprenda comandos para manipulação de dados em SGBD com o PHP

Resumo DevMan

O mercado tecnológico do ponto de vista de armazenamento de dados está cada vez mais avançado. Há uma variedade enorme de bancos de dados disponíveis no mercado e com isso aumentam as chances de termos que portar nossas aplicações para cada um deles. A premissa básica hoje para qualquer profissional de desenvolvimento, é conhecer diversos bancos de dados e suas principais diferenças, sejam elas técnicas ou do ponto de vista de recursos, ou seja, conhecer o SGBD em que se vai trabalhar é extremamente necessário nos dias de hoje.

Veremos nesse artigo como conectar-se a dois bancos de dados bem conhecidos. Conheceremos as principais diferenças em termos de programação entre cada um. Faremos com que nossa aplicação se adapte a cada um.

Nesse artigo veremos

- Comandos necessários para acesso e manipulação de dados em SGBD's;
- Técnicas de adaptação de sistema a mais de um banco de dados;

Qual a finalidade

- Veremos nesse artigo algumas das principais técnicas de acesso a dados em PHP;

Quais situações utilizam esses recursos?

- A principal situação em que podemos aplicar essas técnicas consiste em poder compatibilizar nossos sistemas a mais de um banco de dados, poupando tempo na hora de adequação do mesmo;

Fabrício Desbessel

(fabricio@fabricio.pro.br)

é professor de Linguagem de Programação do Curso Técnico em Informática do Centro Tecnológico Frederico Jorge Logemann de Horizontina/RS e da FAHOR Faculdade Horizontina. Delphiano de coração está sempre disposto a provar que com o Delphi sempre teremos a melhor solução, até mesmo como PHP. Site www.fabricio.pro.br.

O *PHP* surgiu para simplificar ao máximo o desenvolvimento de aplicações WEB. Isso é claramente perceptível quando é necessário escrever um código para, por exemplo, buscar informações em um banco de dados. Com três comandos básicos acessamos o banco, fazemos a consulta e preparamos os dados para exibição.

Nesse artigo vamos aprender esses comandos utilizando dois Sistemas Gerenciadores de Bancos de Dados (SGBD): *MySQL* e *Firebird*. Vamos criar uma base de dados com uma tabela de clientes nos dois bancos de dados e manipular nossa conexão através de comandos e macetes disponíveis no *PHP*.

Ao final desse artigo você estará apto

Clube101.indb 60 10.11.08 15:57:33

a fazer consultas, inserções, alterações e exclusões no banco de dados escrevendo os comandos necessários, sem o auxílio de geradores de aplicação ou ferramentas de *webdesign*.

Criando o banco de dados no MySQL

Para a criação do banco de dados você deve ter instalado o *MySQL Server* (baixe em *dev.mysql.com*) e o *MySQL Administrador* (*dev.mysql.com/downloads/gui-tools/index.html*).

Para criar o banco de dados abra o *MySQL Administrator*, e faça o *login* em *localhost*. Clique em *Catalogs* e em *Schema* clique com o botão direito e escolha *Create New Schema* (**Figura 1**). Você também pode utilizar o atalho das teclas *CTRL+N*. No diálogo *Create new Schema*, informe *direto* para o campo *Schema name*. Após a criação do banco de dados devemos executar o *script* SQL para criação das tabelas e chaves. Para tanto, acesse o menu *Tools>MySQL Command Line Client*. Digite: "use direto;" e pressione ENTER.

Copie o código da **Listagem 1** e digite-o no *command*. Finalize pressionando ENTER novamente, para que as tabelas sejam criadas.

Criando o banco de dados no Firebird

Para a criação do banco de dados você deve ter instalado o *Firebird Server* (baixe em *http://www.firebirdsql.org/*) e, também, algum software para gerenciamento, como o gratuito *IBOConsole* que pode ser encontrado em *www.mengoni.it/Downloads/*. O interessante é que o *IBOConsole* tem tradução para a Língua Portuguesa. Basta acessar o menu *Console > Options* e escolher a opção *Language*. Os passos abaixo utilizarão o *IBOConsole* mas você pode utilizar outro gerenciador a sua escolha.

A primeira coisa a se fazer com o *IBOConsole* é conectar no banco de dados local. Se você acabou de instalar o *Firebird* e o *IBOConsole* é necessário acessar o menu *Servidor > Registrar*. Na tela exibida na **Figura 2** marque a opção *Local Server* e preencha o campo *Descrição*. Depois basta um clique duplo para fazer o login no banco. O nome de super-usuário no *Firebird* é *SYSDBA* e a senha é *masterkey*.

Nota: O usuário *SYSDBA* e a senha *masterkey* é um padrão nas instalações dos SGBD's *Firebird* e *Interbase*. É altamente recomendável a troca dessa senha, o que pode ser realizado no *IBOConsole* pelo menu *Servidor > Segurança de Usuários*.

Nota: Caso ocorra o erro *Client library fbclient.dll not found in the path. Please install it to use this functionality* ao tentar registrar o servidor no *IBOConsole*, basta copiar o arquivo *fbclient.dll* do diretório *bin* na pasta de instalação do *Firebird* diretamente para a pasta *C:\Windows\System32* e tentar novamente.

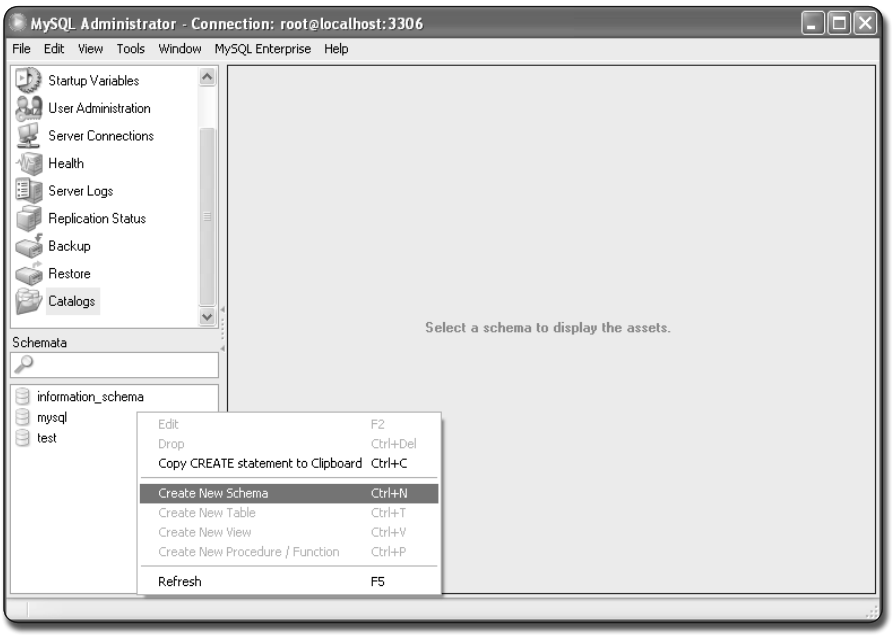


Figura 1. Criação do novo banco no MySQL Administrator

Listagem 1. Script para criação das tabelas no MySQL

```
CREATE TABLE 'clientes' (
  'id' int(10) unsigned NOT NULL auto_increment,
  'nome' varchar(45) NOT NULL,
  'datanasc' date NOT NULL,
  'fone' varchar(10) default NULL,
  'email' varchar(100) default NULL,
  PRIMARY KEY  ('id')
);
```

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora mesmo o portal do assinante ClubeDelphi e assista as aulas de Fabrício Desbessel que ensinam a instalar o banco de dados MySQL Server 5.0 e as ferramentas de administração.

www.devmedia.com.br/articles/viewcomp.asp?comp=4985

Figura 2. Registrando um banco de dados Firebird instalado localmente

Clube101.indb 61 10.11.08 15:57:37

Agora passamos para criação do banco de dados efetivamente. Acesse o menu *Banco de Dados > Criar Banco de Dados*. A primeira definição a ser realizada é do *File Name* onde, deve-se preencher com o caminho e nome do arquivo. Coloque, por exemplo: *C:\Direto.FDB*.

Depois no campo *Size (Pages)* digite o valor 4096 para definir o tamanho de paginação. No campo *SQL Dialect* certifique-se que está o valor 3 (três) pois , nesse dialeto é possível utilizar campos do tipo *date*. Preencha o campo *Alias* e pressione OK (**Figura 3**).

Após esse procedimento o banco de dados já estará criado e aberto, faltando ainda a criação da tabela de clientes. Clique no botão *SQL Interativo* (símbolo SQL) e na aba SQL digite o código contido na **Listagem 2**.

A **Listagem 2** tem o mesmo objetivo da **Listagem 1**, porém, no *Firebird*, precisamos de mais artifícios para criar um campo que seja auto incremento. É necessário criar um *Generator* que funciona como uma variável que conterá o último código utilizado. Para efetuar o incremento utiliza-se uma *Trigger* (gatilho) que é disparado quando houve uma inserção de registro na tabela e tem a função de preencher o campo com o novo valor obtido pela função *GEN_ID* que nada mais é que buscar o valor do *generator* atual e somar mais um.

Conectando ao banco de Dados

Para efetuar uma conexão ao banco de dados precisamos somente de um comando o *connect*. Se for conectar no banco de dados *MySQL* colocamos *mysql_connect*. Se for no Firebird utilizamos o *ibase_connect*. Agora vamos ver a sintaxe completa do *connect*:

Mysql:

```
mysql_connect(SERVIDOR, USUARIO, SENHA);
```

Exemplo: mysql_connect('localhost', 'root', 'root');

Firebird:

```
ibase_connect(SERVIDOR E PATH, USUARIO, SENHA);
```

Exemplo: ibase_connect('localhost:C:\Direto.FDB','SYSDBA','masterkey');

Tanto *mysql_connect* como *ibase_connect* retornam um link com a conexão se a mesma for bem sucedida ou *FALSE* caso ocorra um erro.

Uma diferença perceptível nos comandos é a necessidade de especificar o banco de dados para acesso no *Firebird*, durante a conexão. No *MySQL* essa especificação é realizada somente no momento de executar uma SQL, com um comando específico, explicado posteriormente.

No código da **Listagem 3** tem-se um pequeno script que tenta fazer a conexão com os bancos de dados criados. Para digitar esse código você pode utilizar um editor de texto simples. Eu particularmente digitei no *Delphi For PHP*, pois fica mais fácil de *debugar* e testar o mesmo.

Nota: Se você estiver utilizando *Windows* e um servidor *Firebird*, poderá acontecer um erro ao tentar conectar ao banco de dados: *Failed to locate host machine. Undefined service gds_db/tcp.*

Você terá que abrir o arquivo *C:\WINDOWS\system32\drivers\etc\services* em um editor de texto e adicionar o registro do servidor *Firebird* na porta 3050, conforme o texto abaixo:

gds_db 3050/tcp # Firebird

No código da **Listagem 3** atribui-se os resultados das funções *connect* a uma variável para verificar se a conexão teve sucesso ou não, o que é realizado através dos *if's*.

Listagem 2. Script para criação das tabelas no Firebird

```
CREATE TABLE 'CLIENTES'
(
  'ID' INTEGER NOT NULL,
  'NOME' VARCHAR(45) NOT NULL,
  'DATANASC' DATE NOT NULL,
  'FONE' VARCHAR(10),
  'EMAIL' VARCHAR(100),
 PRIMARY KEY ('ID')
);

CREATE GENERATOR 'GEN_ID_CLIENTE';
SET TERM ^ ;

CREATE TRIGGER 'TG_CLIENTES' FOR 'CLIENTES'
ACTIVE BEFORE INSERT POSITION 0
AS
begin
    NEW.ID= GEN_ID(gen_id_cliente, 1);
end
 ^
```

Listagem 3. Script PHP para testes de conexão aos bancos de dados

```
<?php
    $MYSQL = mysql_connect('localhost','root','root');
    $FIREBIRD = ibase_connect('localhost:C:\Direto.FDB','SYSDBA','masterkey');

    echo "<BR>";
    if ($MYSQL) {
      echo "MYSQL - CONECTADO";
    };
    echo "<BR>";
    if ($FIREBIRD) {
      echo "FIREBIRD - CONECTADO";
    };

    mysql_close($MYSQL);
    ibase_close($FIREBIRD);
?>
```

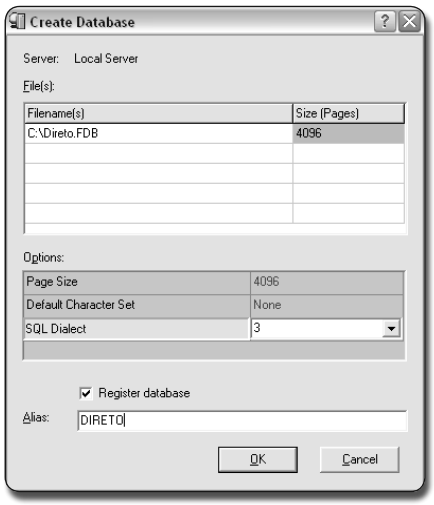


Figura 3. Criação do banco de dados Firebird no IBOConsole

Clube101.indb 62 10.11.08 15:57:39

Também utilizamos os comandos *mysql_close* e *ibase_close* para fecharmos as conexões, passando como parâmetro as variáveis criadas para atribuição do link da conexão, e, se houver falha essa variável terá como valor *FALSE*. O *close* é um comando opcional uma vez que o PHP libera todos os recursos no fechamento do Script.

É importante ressaltar a necessidade de fecharmos as conexões, não correndo o risco de esgotar o número de conexões simultâneas (parâmetro do *PHP* no arquivo *PHP.ini*), bem como recursos do servidor, quando, após um acesso ao banco de dados, você tem bastante código *PHP* para executar ainda. Principalmente se sua conexão não for persistente.

Listagem 4. Script com informações necessárias para conexões ao banco de dados

```
<?php
   $MServidor = "localhost";
   $MUsuario = "root";
   $MSenha = "root";

   $FServidor = "localhost:C:\Direto.FDB";
   $FUsuario = "SYSDBA";
   $FSenha = "masterkey";
?>
```

Listagem 5. Script para teste de conexão

```
<?php
    require_once('conexao.php');
    $MYSQL = mysql_pconnect($MServidor,$MUsuario,$MSenha);
    $FIREBIRD = ibase_pconnect($FServidor,$FUsuario,$FSenha);

    echo "<BR>";
    if ($MYSQL) {
      echo "MYSQL - CONECTADO";
    };
    echo "<BR>";
    if ($FIREBIRD) {
      echo "FIREBIRD - CONECTADO";
    };
?>
```

Conexões persistentes ao banco de dados

Tanto *MySQL* como *Firebird* possuem um comando para criar conexões persistentes ao banco de dados. O uso de conexões persistentes pode ser interessante em caso de muitos acessos simultâneos ao banco de dados. O que acontece com esse tipo de conexão é o reaproveitamento, uma vez que, a cada nova conexão, o sistema primeiro verifica se já não há uma conexão ao mesmo banco, com o mesmo usuário e senha. Então se você tem vários processos sendo executados ao mesmo tempo, todos poderão compartilhar uma única conexão ao banco de dados.

Um porém, nesse tipo de conexão é a dependência da forma como o *PHP* é executado no servidor. Se for como um *CGI* essa persistência não funciona, pois, a cada requisição as conexões serão instanciadas e no final, destruídas. Agora se o *PHP* funcionar como um módulo (grande maioria) você poderá ter muita eficiência nas conexões usando a persistência, pois depois da primeira conexão a sua reutilização será muito mais rápida que criar uma nova.

Usando a persistência o comando *close* (*mysql_close* e *ibase_close*) não irá mais fechar a conexão, pois a idéia é manter a mesma sempre ativa. Aí abre-se um precedente para termos alguns problemas quando precisamos travar algumas tabelas para edição, pois para destravar é necessário fechar a conexão.

O que muda para realizarmos uma conexão persistente? Somente a adição da letra "**p**" ao comando *connect*.

Mysql:

```
mysql_pconnect(SERVIDOR, USUARIO, SENHA);
```

Exemplo: mysql_pconnect('localhost', 'root', 'root');

Firebird:

```
ibase_pconnect(SERVIDOR E PATH, USUARIO, SENHA);
```

Exemplo: ibase_pconnect('localhost:C:\Direto.FDB','SYSDBA','masterkey');

Tente agora modificar o código da **Listagem 3** tornando suas conexões persistentes.

Arquivo com a definição de banco de dados

Agora pense você em uma aplicação que tenha inúmeros scripts *PHP* que fazem conexão ao banco e precisam das informações de servidor, usuário e senha. Você não poderá escrever essas informações em todos os arquivos. Precisa-se criar um arquivo para centralizar essas informações em um único local, facilitando assim a troca em qualquer momento.

Por isso é normal criar um arquivo *PHP* que contenha essas informações. Para exemplificar tem-se o código da **Listagem 4** onde estão as informações para nossas conexões e no código da **Listagem 5** tem-se o teste de conexão, buscando as informações do outro arquivo.

No código da **Listagem 4** simplesmente declarou-se variáveis (iniciam com **$**), definindo como seus valores, as informações necessárias para conectar no banco de dados. No código da **Listagem 5** utilizamos essas variáveis para executar o *connect*. O comando *require_once* serve para incluir o arquivo especificado entre parêntese no *script*. Em outras palavras o *require_once* junta os arquivos na hora de execução criando um único. Também, com a mesma idéia existe o comando *require* que difere pela não observância de o arquivo já ter sido incluído podendo ocorrer problemas, como redefinição de funções, valores de variáveis, etc. Prefira sempre utilizar o *require_once*.

Nota do DevMan

Tente utilizar suas conexões de forma persistente. Só opte por conexões não persistentes quando for necessário travar uma tabela ou ter blocos de transações que dependem um do outro. Utilizando conexões persistentes suas aplicações terão uma performance melhor.

Clube101.indb 63 10.11.08 15:57:43

Listagem 6. Script para localização de dados

```
<?php
    require_once('conexao.php');
    $MYSQL = mysql_pconnect($MServidor,$MUsuario,$MSenha);
    $FIREBIRD = ibase_pconnect($FServidor,$FUsuario,$FSenha);

    mysql_select_db('direto',$MYSQL);
    $SQL = "insert into clientes (id, nome, datanasc, fone, email)
      values (0,'Fabrício Desbessel', '1997-03-17', '1135122020','fabricio@fabricio.pro.br')";
    $MResultado = mysql_query($SQL, $MYSQL);
    $FResultado = ibase_query($SQL, $FIREBIRD);

    echo "<BR>";
    if ($MResultado) {
      echo "MYSQL - Dado inserido com sucesso";
    };
    echo "<BR>";
    if ($FResultado) {
      echo "FIREBIRD - Dado inserido com sucesso";
    };
?>
```

Listagem 7. Script para alteração de dados

```
<?php
    require_once('conexao.php');
    $MYSQL = mysql_pconnect($MServidor,$MUsuario,$MSenha);
    $FIREBIRD = ibase_pconnect($FServidor,$FUsuario,$FSenha);

    mysql_select_db('direto',$MYSQL);
    $SQL = "update clientes set nome='Fabrício Clube Delphi' where id=1";
    $MResultado = mysql_query($SQL, $MYSQL) or die(mysql_error());
    $FResultado = ibase_query($SQL, $FIREBIRD) or die(ibase_errmsg());

    echo "<BR>";
    if ($MResultado) {
      echo "MYSQL - Dado alterado com sucesso";
    };
    echo "<BR>";
    if ($FResultado) {
      echo "FIREBIRD - Dado alterado com sucesso";
    };
?>
```

Listagem 8. Script para exclusão de dados

```
<?php
    require_once('conexao.php');
    $MYSQL = mysql_pconnect($MServidor,$MUsuario,$MSenha);
    $FIREBIRD = ibase_pconnect($FServidor,$FUsuario,$FSenha);

    mysql_select_db('direto',$MYSQL);
    $SQL = "delete from clientes where id=1";
    $MResultado = mysql_query($SQL, $MYSQL) or die(mysql_error());
    $FResultado = ibase_query($SQL, $FIREBIRD) or die(ibase_errmsg());

    echo "<BR>";
    if ($MResultado) {
      echo "MYSQL - Dado excluído com sucesso";
    };
    echo "<BR>";
    if ($FResultado) {
      echo "FIREBIRD - Dado excluído com sucesso";
    };
?>
```

Para incluir um script PHP dentro de outro podemos também utilizar os comandos *include* e *include_once*. Eles funcionam da mesma forma que o *require* e *require_once*, porém se acontecer um erro, como, por exemplo, o arquivo não ser encontrado, ou até mesmo erro no arquivo a ser incluído, o *include* irá somente gerar uma mensagem de *Warning* (cuidado) executando o restante do *script*. Já o *require*, em caso de erro produzirá um *Fatal error* (erro fatal) impedindo a execução do script restante.

Portanto, em caso de conexão a banco de dados é muito interessante utilizar o *require* para incluir os valores necessários para conectar, pois, se acontecer um erro, devemos abordar a execução do *script*.

Manipulação de Dados

Para inserir, alterar ou excluir dados utiliza-se instruções SQL. Para executá-las utilizamos um comando *query* diferenciando o início do comando conforme o banco de dados utilizado.

Mysql:

```
mysql_query(INSTRUÇÃO SQL, CONEXÃO);
```

Exemplo: mysql_query("insert into clientes (id, nome, datanasc, fone, email) values (0,'Fabrício Desbessel', '1997-03-17', '1135122020','fabricio@fabricio.pro.br')", $MYSQL);

Firebird:

```
ibase_query(INSTRUÇÃO SQL, CONEXÃO);
```

Exemplo: ibase_query("insert into clientes (id, nome, datanasc, fone, email) values (0,'Fabrício Desbessel', '1997-03-17', '1135122020','fabricio@fabricio.pro.br')", $FIREBIRD);

Os comandos *mysql_query* e *ibase_query* possuem a mesma funcionalidade que é executar uma instrução SQL no banco de dados retornando um valor booleano conforme o sucesso da operação. Para testar digite e execute o código da **Listagem 6.**

O código da **Listagem 5** inicia com a inclusão do arquivo *conexao.php* que contém as informações necessárias para a conexão ao banco de dados. Em seguida são realizadas as conexões persistentes ao banco de dados *MySQL* e *Firebird*.

Clube101.indb 64 10.11.08 15:57:47

O comando *mysql_select_db* serve para selecionar o banco de dados, que no exemplo tem o nome *"direto"*, utilizando, como segundo parâmetro a variável que contém o link da conexão ao banco. Esse comando não é necessário no *Firebird* porque na conexão já informa-se o nome do banco, através do nome do arquivo *.FDB*. Se você não utilizar esse comando o *MySQL* vai executar a SQL no banco de dados utilizado na última vez.

Para executar a *query* e saber se a mesma foi executada com sucesso atribuímos o seu retorno a uma variável que irá receber *TRUE* ou *FALSE*. Pode-se ainda mandar parar a execução caso aconteça algum erro, com o uso do *die* que é um comando de *exit* onde é possível exibir a mensagem de erro. Veja como ficaria o comando utilizando o *die*:

```
$MResultado = mysql_query($SQL, $MYSQL) or
  die(mysql_error());
$FResultado = ibase_query($SQL, $FIREBIRD) or
  die(ibase_errmsg ());
```

Utilizando *or die* o comando terá que ser executado com sucesso ou será executado um *exit*, abandonando a execução dos comandos restantes no script e ainda exibindo a mensagem de erro que é apresentada pelas funções *mysql_error* e *ibase_errmsg*.

Como todos SGBD's utilizam o padrão ANSI para SQL é possível escrever da mesma forma a instrução SQL o que resultou em uma única instrução no código da **Listagem 5** que foi armazenado em uma variável ($SQL) para ser reaproveitada.

Da mesma forma pode-se executar instruções SQL de *Update* e *Delete*, como é demonstrado nos códigos das **Listagens 7** e **8**, respectivamente.

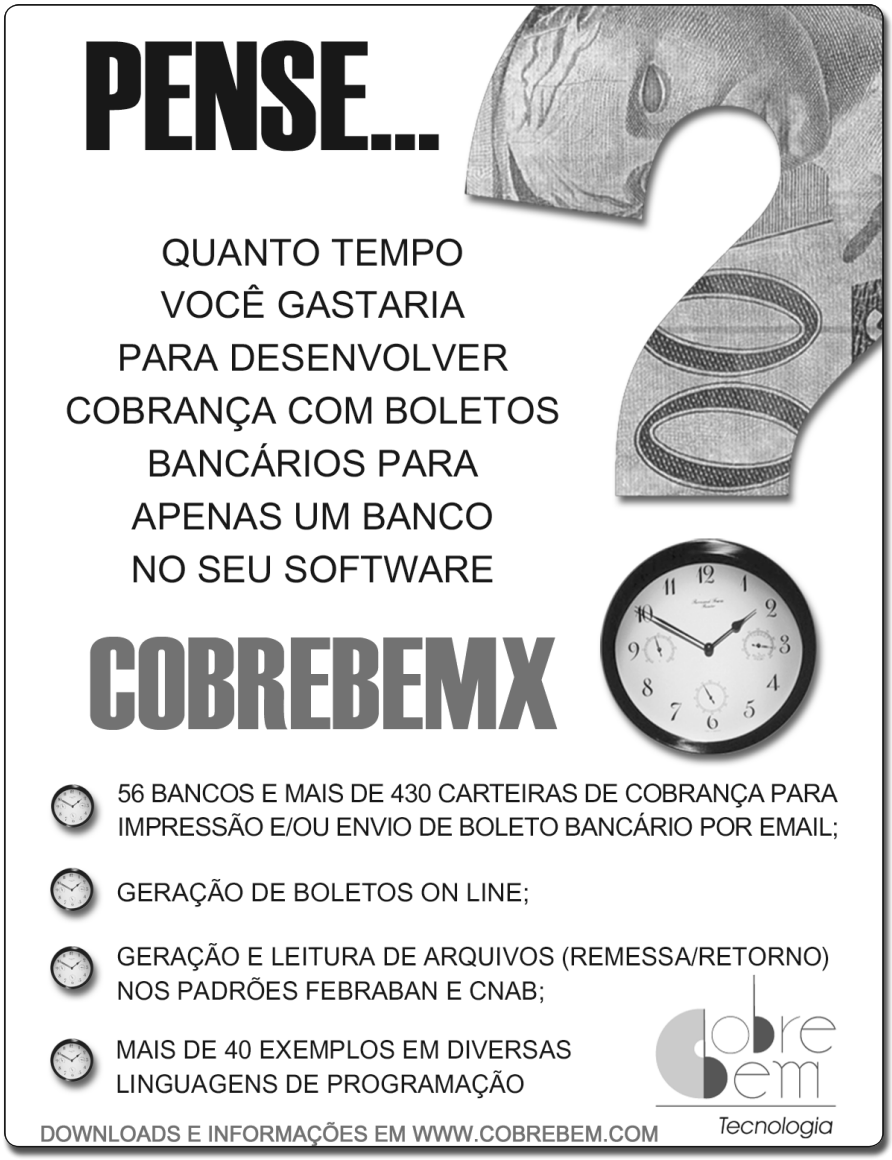
Para qualquer manipulação de dados pode-se utilizar o *query*, o que pode-se considerar muito positivo, pois facilita a criação de funções genéricas para abstrair as instruções SQL e até mesmo as conexões a banco de dados diferentes.

Seleção de Dados - Consultas

Para executar instruções SQL de *Select* utiliza-se também o comando *query*. Porém, faz-se necessário buscar linha por linha resultante do *select* associando os resultados em uma matriz, através do

Listagem 9. Script para consulta no banco de dados

```
<?php
    require_once('conexao.php');
    $MYSQL = mysql_pconnect($MServidor,$MUsuario,$MSenha);
    $FIREBIRD = ibase_pconnect($FServidor,$FUsuario,$FSenha);
    mysql_select_db('direto',$MYSQL);
    $SQL = "select * from clientes order by nome";
    $MResultado = mysql_query($SQL, $MYSQL);
    $FResultado = ibase_query($SQL, $FIREBIRD);
    echo "<BR>";
    echo "MYSQL - Dados selecionados com sucesso:<BR>";
    while ($MColuna = mysql_fetch_assoc($MResultado)) {
      echo $MColuna["id"]." - ";
      echo $MColuna["nome"]." - ";
      echo $MColuna["datanasc"]." - ";
      echo $MColuna["fone"]." - ";
      echo $MColuna["email"]."<BR>";
    }
    mysql_free_result($MResultado);
    echo "<BR>";
    echo "Firebird - Dados selecionados com sucesso:<BR>";
    while ($FColuna = ibase_fetch_assoc($FResultado)) {
      echo $FColuna["ID"]." - ";
      echo $FColuna["NOME"]." - ";
      echo $FColuna["DATANASC"]." - ";
      echo $FColuna["FONE"]." - ";
      echo $FColuna["EMAIL"]."<BR>";
    }
    ibase_free_result($FResultado);
?>
```



Clube101.indb 65 10.11.08 15:57:52

Listagem 10. Script de consulta otimizada no Firebird

```
<?php
    require_once('conexao.php');
    $FIREBIRD = ibase_pconnect($FServidor,$FUsuario,$FSenha);

    $SQL = "select * from clientes order by nome";
    $Consulta = ibase_prepare($FIREBIRD, $SQL);
    $FResultado = ibase_execute($Consulta);

    echo "<BR>";
    echo "Firebird - Dados selecionados com sucesso:<BR>";
    while ($FColuna = ibase_fetch_assoc($FResultado)) {
      echo $FColuna["id"]." - ";
      echo $FColuna["NOME"]." - ";
      echo $FColuna["DATANASC"]." - ";
      echo $FColuna["FONE"]." - ";
      echo $FColuna["EMAIL"]."<BR>";
    }
    ibase_free_result($FResultado);
?>
```

comando *fetch_assoc*. Veja na **Listagem 9** como fazer uma consulta e exibir as informações resultantes.

No código da **Listagem 8,** executa-se uma consulta em cada banco de dados, e usa-se um laço de repetição *while* para buscar os valores de cada linha resultante da consulta por meio da função *fetch_assoc* que retorna uma matriz com as informações da linha. Para acessar o valor de uma coluna, a forma mais simples é utilizar a variável passando como índice o nome da coluna. Como é possível em uma consulta ter nomes iguais para as colunas podemos usar a função *fetch_row* que faz o mesmo da função *fetch_assoc* só que como índice utiliza valores numéricos, na mesma ordem dos campos selecionados através da cláusula *select*. Assim pode-se utilizar, por exemplo *$FColuna[1]* para buscar a informação do campo 1.

Existe uma diferença significativa entre *MySQL* e *Firebird* na hora de exibir as informações. No *Firebird* foi necessário colocar o nome dos campos em maiúsculo. Isso porque o script de criação das tabelas (**Listagem 2**) estava todo maiúsculo. Para o PHP isso é muito significativo e sugere-se cuidar bastante a nomeação, tanto de campos como de nome de tabelas no banco de dados.

Ao final da execução da consulta faz-se necessário liberar a memória utilizada para guardar as informações resultantes, função do comando *free_result*. Acho que não é necessário, mas vou ressaltar: não pode-se esquecer de liberar a memória. Imagina você 100 mil registros selecionados em uma consulta e o desenvolvedor esquecer de liberar. Assim não tem servidor com recursos suficientes! Mesmo que depois da execução do *script* tudo seja liberado da memória isso é essencial, pois mesmo em frações de segundos, você pode esgotar os recursos do servidor, impedindo a execução com sucesso de outro script e, até mesmo, ter problemas no seu.

Otimizando consultas no Firebird

Quem conhece e utiliza o *Firebird* sabe que o mesmo possui formas de preparar as SQL e depois executá-las. Há um ganho muito grande de performance na preparação de uma consulta quando a mesma é utilizada repetidamente, o que acontece bastante em sites com muitos acessos. Essa otimização acontece porque o *Firebird* monta um plano de execução para a consulta e esse plano poderá ser reutilizado para uma consulta igual, diminuindo o tempo total de execução da consulta.

Na **Listagem 10** fazemos uma consulta no *Firebird* utilizando as funções *prepare* e *execute*, que proporcionam mais velocidade e otimização de recursos.

Olhando o código da **Listagem 10,** verifica-se o funcionamento das funções *prepare* e *execute* que são específicas do *Firebird*. No *prepare,* passa-se como parâmetro a conexão com o banco e a instrução SQL. Com isso o *Firebird* já cria o plano de execução da consulta. O *execute* necessita, como parâmetro, somente a variável que recebeu o retorno do *prepare*. O restante do código é igual, o que torna a utilização do *prepare* e *execute* uma obrigação em *script* que conectam ao *Firebird*, pois necessita pequenas modificações no código (duas linhas) e agiliza muito a consulta e, conseqüentemente, a execução do *script*.

Conclusão

Realmente é muito fácil manipular dados no *PHP*. Com apenas alguns comandos faz-se todas as operações necessárias em uma aplicação.

Também fica muito fácil você mesmo criar suas funções como, conecta, insere, altera, consulta, etc. Ainda melhor, é, se essas funções abstraírem a conexão com o banco de dados. Assim sua aplicação poderia ser multi-banco, o que seria um grande diferencial.

Agora está na hora de testar seu aprendizado. Coloque a mão na massa e crie suas próprias funções. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Clube101.indb 66 10.11.08 15:58:01