Conectando PHP ao banco de dados

Fale com o banco

Usando PHP-GTK, é fácil escrever programas gráficos que se comunicam com bancos de dados.

por Pablo Dall'Oglio

No artigo anterior[1] estudamos as formas de se reagir às ações do usuário através da programação de sinais. Neste quarto artigo, estudaremos a conexão a bancos de dados. Em princípio, a conexão a bancos de dados em *PHP-GTK* funciona exatamente da mesma maneira que no ambiente web. A diferença é que, em alguns casos, precisaremos especificar o IP do servidor onde o banco de dados está localizado no momento de conexão, visto que o banco não estará localizado na máquina cliente, que é onde o PHP-GTK está rodando. Aqui, mostraremos a utilização do PHP-GTK juntamente com o *SQLite*, um banco de dados em forma de arquivo que dispensa a instalação de servidor, permitindo que toda a aplicação rode na máquina cliente.

SQLite

O SQLite é um banco de dados relacional cuja estrutura (tabelas, índices, dados) está contida em um único arquivo. O acesso aos dados é implementado por uma biblioteca de funções escritas em C por Richard Hipp, que é parte integrante do *PHP5*, tornando seu uso extremamente simples e em muitas vezes mais veloz que bancos de dados como *PostgreSQL* e *MySQL*. A manipulação dos dados é realizada através da linguagem *SQL*.

O leitor talvez esteja acostumado a bancos de dados relacionais cuja estrutura cliente-servidor exige a instalação do gerenciador de banco de dados, que se comunica com a aplicação, geralmente através de uma porta específica, via protocolo TCP/IP. O SQLite não tem esse comportamento, pois o banco se resume a um único arquivo com a extensão `.db`, que contém todas as tabelas do sistema. Para distribuir uma aplicação que faça uso de um banco de dados SQLite, basta compactar esse arquivo `.db` e distribuí-lo junto à aplicação.

O SQLite funciona de maneira similar aos arquivos *dBase* (`.dbf`) ou *Access* (`.mdb`), proporcionando uma estrutura simples de banco de dados em forma arquivo; porém, o SQLite é bem mais robusto que seus antecessores, tendo em vista que implementa o padrão SQL92, permite transações, *triggers* e bancos de dados de até 2 TiB de tamanho, ou seja, deve ser suficiente para a grande maioria dos usos. O banco de dados pode servir várias requisições de leitura (`SELECT`) ao mesmo tempo. Entretanto, o arquivo é bloqueado com um *lock* em operações de escrita (`INSERT`, `UPDATE`, `DELETE`).

Deve-se ressaltar que a utilização do SQLite é recomendada para ambientes onde existe pouca concorrência. Por isso, ele é indicado para o ambiente desktop. Em ambiente web, onde temos vários usuários tentando realizar transações com o banco de dados, seu desempenho provavelmente será bem inferior ao dos tradicionais bancos cliente-servidor.

Criando o banco de dados

A criação de um banco de dados SQLite é extremamente simples. Existe uma ferramenta de administração do banco de dados em linha de comando, chamada *sqlite2* ou *sqlite3*, dependendo da versão utilizada, disponível no site do SQLite para várias plataformas. O banco de dados também poderá ser criado pelo próprio PHP, por meio da função `sqlite_open()`. Essa função é responsável por abrir a conexão a um

Exemplo 1: Criação da tabela

```
01 <?php
02 // abre|cria o banco de dados
03
04 $db = sqlite_open('meubanco.db');
05
06 $sql = 'create table pessoas (' .
07        ' id integer, ' .
08        ' nome varchar(80), ' .
09        ' email varchar(80) ) ';
10
11 // cria tabela
12 sqlite_query($db, $sql);
13 // fecha conexão
14 sqlite_close($db);
15 ?>
```

Figura 1 A criação de interfaces *GTK* com o *Glade* é muito prática.

banco de dados ou, caso ele ainda não exista, criá-lo. No **exemplo 1**, criamos uma tabela chamada `pessoas`, com colunas para código, nome e email.

Criando a interface

O próximo passo é criar a interface da aplicação. Poderíamos criar a janela e os objetos manualmente, mas vamos aproveitar as facilidades que o *Glade* nos oferece. O Glade é uma ferramenta utilizada para criar o visual da aplicação, e funciona da mesma maneira tanto no Linux quanto no Windows®, como vimos no segundo artigo desta série[2].

Para criar nossa interface (**figura 1**) vamos utilizar primeiramente o componente `GtkWindow`, que irá criar uma janela. Após isso, vamos colocar um `GtkFrame` com o título *Cadastro de Pessoas* dentro da janela. Então, dentro desse *frame*, vamos colocar um `GtkFixed`, que permite ancorarmos nossos objetos em coordenadas absolutas. Vamos criar três `GtkLabel` (*Código*, *Nome* e *Email*) e três `GtkEntry` para a digitação de valores. Observe na **figura1** que, ao criarmos os objetos `GtkEntry`, damos nomes a eles pela janela *Propriedades*. A janela de propriedades é utilizada também para regular tamanhos e posições dos objetos em tela. É importante utilizarmos nomes que serão facilmente lembrados posteriormente para que possamos capturar esses objetos na aplicação.

Criando o programa

Após criarmos a interface no passo anterior, salvaremos o arquivo com o nome `tela.glade`. Esse arquivo será utilizado agora em nossa aplicação. Não precisaremos criar a interface no código-fonte, mas sim lembrarmos dos nomes que atribuímos aos objetos no Glade (pela aba *Propriedades*), para que possamos capturar esses objetos. No **exemplo 2**, a classe `GladeXML` realiza a leitura do arquivo `.glade` e disponibiliza o método `get_widget()`. Esse método captura os objetos "desenhados" pelo Glade e disponibiliza-os na aplicação como objetos normais do GTK. A partir daí, só temos que capturá-los utilizando o seu nome. Criamos três objetos `GtkEntry` (`codigo`, `nome`, `email`) e um botão chamado *Salvar*. Nesse programa, estamos conectando o botão *Salvar* à função `onSave()`. Sempre que o usuário clicar no botão *Salvar*, essa função será executada, lendo os valores que foram digitados nos objetos `GtkEntry` e inserindo-os na tabela *pessoas* do banco de dados `meubanco.db`. A **figura 2** mostra o programa rodando e um registro sendo inserido no banco de dados. Na seqüência, temos o código-fonte desse exemplo.

Figura 2 Execução do programa escrito no **exemplo 2**.

Exemplo 2: Lógica do programa para inserção de dados no banco

```
01 <?php
02 // lê objeto glade
03 $glade=new GladeXML('tela.glade');
04 // captura objetos
05 $codigo=$glade->get_widget('codigo');
06 $nome  =$glade->get_widget('nome');
07 $email =$glade->get_widget('email');
08 $salvar=$glade->get_widget('salvar');
09 // define a ação do botão
10 $salvar->connect('clicked','onSave');
11 /* função onSave
12  * armazena os dados digitados
13  * no banco de dados */
14 function onSave()
15 {
16     global $codigo,$nome,$email;
17     // obtém valores dos objetos
18     $valor1 = $codigo->get_text();
19     $valor2 = $nome->get_text();
20     $valor3 = $email->get_text();
21     // abre conexão com o banco
22     $db=sqlite_open('meubanco.db');
23     // cria a string SQL
24     $sql="insert into pessoas values (".
25      "'$valor1','$valor2','$valor3')";
26     // executa o comando
27     sqlite_query($db, $sql);
28     // fecha conexão
29     sqlite_close($db);
30     // cria um diálogo de mensagem
31     $dialog=new GtkMessageDialog(
32         null, Gtk::DIALOG_MODAL,
33               Gtk::MESSAGE_INFO,
34               Gtk::BUTTONS_OK,
35         'Registro inserido !!');
36     // exibe o diálogo
37     $dialog->run();
38     // destrói o diálogo
39     $dialog->destroy();
40 }
41 Gtk::Main();
42 ?>
```

Conclusão

Neste artigo, vimos como armazenar as informações de nossa aplicação no banco de dados por meio de um formulário de entrada de dados. No próximo e último artigo da série, iremos estudar a interoperabilidade entre aplicações através da utilização de *web services*. ■

Mais informações

[1] "Programação de sinais", Linux Magazine 39: http://www.linuxmagazine.com.br/article/programacao_de_sinais

[2] "Hora de Construir", Linux Magazine 38: http://www.linuxmagazine.com.br/article/hora_de_construir

[3] PHPO-GTK Brasil: http://www.php-gtk.com.br

[4] Livro PHP-GTK: http://www.php-gtk.com.br/book

[5] Site do autor: http://www.pablo.blog.br

[6] SQLite: http://www.sqlite.org