por Pablo Dall'Oglio

Criando uma Agenda com PHP-GTK e SQLite

Neste artigo iremos construir uma ferramenta para gestão de compromissos em PHP-GTK utilizando o banco de dados SQLite para armazenar os dados.

Como a maioria de vocês já sabe, o PHP conecta em todos os bancos de dados importantes existentes (Postgres, Mysql, Oracle, SqlServer, Firebird, DB2, dentre outros). Na biblioteca GTK programamos o acesso ao banco de dados da mesma forma que uma aplicação web, com a diferença de que o acesso aos dados será remoto (cliente-servidor).

Já há algum tempo eu pensava em um exemplo bem simples de aplicação em PHP-GTK, em que as pessoas pudessem copiar e colar e já sair utilizando conjuntamente com um banco de dados, mas, para isso, o banco de dados teria de ser de fácil instalação.

Foi aí que me veio a idéia de desenvolver uma Agenda, afinal, todos nós precisamos controlar nossos compromissos. E resolvi utilizar o SQLite, um banco de dados em sistema de arquivos que poderia ser compactado junto com a aplicação, dispensando a instalação de um servidor de banco de dados, tornando sua instalação infinitamente simples.

1. SQLite

O SQLite é um banco de dados relacional cuja estrutura (tabelas, índices, dados) está contida em um único arquivo no sistema. O acesso aos dados é implementado por uma biblioteca de funções escritas em C por Richard Hipp e a manipulação dos dados é realizada por meio da linguagem SQL.

Você deve estar acostumado com bancos de dados relacionais cuja estrutura cliente-servidor exige a instalação do servidor de banco de dados, que irá se comunicar com a aplicação, geralmente através de uma porta específica, via protocolo TCP/IP. Como o SQLite não tem nada disto, ele pode ser compactado juntamente com a aplicação. Imagine um arquivo chamado "meusistema.db" contendo todas as tabelas do seu sistema! Você só terá de compactar este arquivo junto com sua aplicação para distribuir seu programa! Lembra do formato .DBF, comum entre as aplicações clipper, ou os arquivos .MDB do MS Access ? Pois é, o SQLite tem o mesmo foco: proporcionar uma estrutura de banco de dados simples em arquivo para ser distribuído juntamente com aplicações *standalone*, mas é muito melhor que seus antecessores, tendo em vista que implementa o padrão SQL92, permite transações, *triggers* e permite bancos de dados de até 2 tebibytes de tamanho (2 na 41), ou seja, é muito byte. O banco de dados pode servir várias requisições de leitura (SELECT) ao mesmo tempo, entretanto, um lock do arquivo é realizado em operações de escrita (INSERT, UPDATE, DELETE).

A biblioteca de acesso ao SQLite é parte integrante do PHP5. Isto quer dizer, que as funções de criação e acesso ao banco de dados são nativas da linguagem, o que torna seu uso extremamente simples.

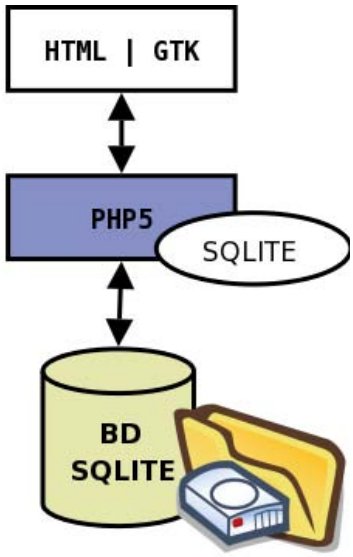

Figura 1 – Estrutura do SQLite

2. Classe Agenda

Para começar nosso programa, criaremos uma classe chamada Agenda. Iremos criá-la utilizando o mecanismo de herança. Agenda será classe filha de GtkWindow, logo, ela também será uma janela e irá possuir todos os métodos que esta possui. No método construtor, além de instanciar e centralizar a janela, iremos criar um componente calendário e conectar dois de seus sinais: *day-selected*, que é disparado sempre que é selecionada uma data, e *month-changed*, que é disparado sempre que o usuário visualiza outro mês. Iremos criar campos para digitação de data, hora, título e descrição da tarefa. Também iremos criar dois botões de ação: um para salvar o compromisso digitado e outro para excluir um compromisso da listagem de compromissos. Esta listagem será criada pelo método createTaskList(), que irá criar toda estrutura do GtkTreeView para visualizar as tarefas.

```
<?php
/*
 * classe Agenda
 * Pablo Dall'Oglio
 */
class Agenda extends GtkWindow
{
private $calendar;
/*
 * método construtor
 * instancia a janela e posiciona elementos
 */
public function __construct()
{
    // instancia janela
    parent::__construct();
    // define posicionamento
    parent::set_default_size(700,500);
    parent::set_position(GTK::WIN_POS_CENTER)
    // quando for fechada, aborda app
    parent::connect_simple('destroy', array
(Gtk, 'main_quit'));

    // cria Fixed, para posicionar widgets
    $fixed = new GtkFixed;
    // adiciona GtkFixed dentro da janela
    parent::add($fixed);

    // cria objeto calendário
    $this->calendar = new GtkCalendar;
    // conecta sinais de mudança de dia e mês
    $this->calendar->connect_simple('day-
selected', array($this, 'onDaySelected'));

    $this->calendar->connect_simple('month-
changed', array($this, 'onMonthChanged'));
    $fixed->put($this->calendar, 10,10);

    // cria campos de digitação para
    // data, hora, título e descrição
    $this->data   = new GtkEntry;
    $this->hora   = new GtkEntry;
    $this->titulo = new GtkEntry;
    $this->descr  = new GtkTextView;

    // cria janela de rolagem para descrição
    $scroll = new GtkScrolledWindow;
    $scroll->set_size_request(240,100);
    $scroll->add($this->descr);

    // cria e posiciona rótulos de texto
    $fixed->put(new GtkLabel('Data'),10,200);
    $fixed->put(new GtkLabel('Hora'),140...);
    $fixed->put(new GtkLabel('Título'),...);
    $fixed->put(new GtkLabel('Descrição')..);

    // posiciona campos de digitação
    $fixed->put($this->data,  10, 220);
    $fixed->put($this->hora, 140, 220);
    $fixed->put($this->titulo,10, 270);
    $fixed->put($scroll, 10, 320);

    // cria botão de salvar
    $botao = GtkButton:: new_from_stock
(GTK::STOCK_SAVE);
    // conecta o botão ao método onSave
    $botao->connect_simple('clicked', array
($this, 'onSave'));
    $fixed->put($botao, 160, 440);

    // cria botão de deletar
    $botao = GtkButton:: new_from_stock
(GTK::STOCK_DELETE);
    // conecta o botão ao método onDelete
    $botao->connect_simple('clicked', array
($this, 'onDelete'));
    $fixed->put($botao, 560, 440);

    // cria listagem de tarefas
    $this->createTaskList();

    // coloca listagem dentro de um scroll
    $scroll = new GtkScrolledWindow;
    $scroll->add($this->list);
    $fixed->put($scroll, 280, 10);

    // cria banco de dados
    $this->createDatabase();

    // atualiza lista de tarefas
    $this->onMonthChanged();
}
```

Agora vamos ao método createDatabase(). Este método é executado toda vez que a aplicação inicializa e é responsável por conectar ao banco de dados. Caso o banco de dados não exista, ele também irá criá-lo.

```
/*
 * método createDatabase
 * cria banco de dados, se ele não existir
 */
function createDatabase()
{
    // verifica se existe o banco
    if (!file_exists('agenda.db'))
    {
        // conecta ao banco
        // (cria se não existir)
        $this->conn = new PDO
('sqlite:agenda.db');
        // cria a tabela
        $this->conn->Query('create table agen-
da (data, hora, titulo, descricao)');
    }
    else
  {
        // conecta ao banco
        $this->conn = new PDO
```

```
('sqlite:agenda.db');
    }

}
```

O próximo método que iremos criar é o createTaskList(). Este método irá criar a listagem de tarefas que fica ao lado do calendário.

```
/*
 * método createTaskList
 * cria a listagem de tarefas
 */
function createTaskList()
{
    // cria objeto treeview
    $this->list = new GtkTreeView;
    // cria modelo de dados com 4 posições
    $this->model = new GtkListStore(...);
    $this->list->set_model($this->model);

    // cria 4 colunas
    $column1 = new GtkTreeViewColumn('Data');
    $column2 = new GtkTreeViewColumn('Hora');
    ...

    // cria 4 renderizadores de texto
    $renderer1 = new GtkCellRendererText();
    $renderer2 = new GtkCellRendererText();
    ...

    // empacota os renderizadores
    $column1->pack_start($renderer1, true);
    $column2->pack_start($renderer2, true);
    ...
    // adiciona as colunas na treeview
    $this->list->append_column($column1);
    $this->list->append_column($column2);
    ...

    // define dimensões da treeview
    $this->list->set_size_request(400,400);
}
```

Após o método createTaskList(), temos o método onDaySelected(). Ele é executado sempre que o usuário clica em alguma data do calendário. Seu objetivo é preparar os campos para digitação de um novo compromisso. Ele limpa todos os campos de digitação, preenchendo o campo "data" justamente com a data selecionada.

```
/*
 * método onDaySelected
 * executado sempre que o usuário
 * seleciona outra data
 */
function onDaySelected()
{
    // obtém a data selecionada
    $date = $this->calendar->get_date();
    $ano = $date[0];
    $mes = $date[1] +1;
    $dia = $date[2];

    // limpa os campos de digitação
    $this->data->set_text("$dia/$mes/$ano");
    $this->hora->set_text('');
    $this->titulo->set_text('');
    $buffer = $this->descr->get_buffer();
    $buffer->delete($buffer->get_start_iter(),
$buffer->get_end_iter());
}
```

Nosso próximo método é o onMonthChanged(). Ele será executado sempre que o usuário navegar no calendário, alterando seu mês. Seu objetivo é recarregar todos os compromissos do mês corrente. Desta forma, este método será executado também em outras ocasiões, quando, por exemplo, for inserido um novo registro ou for excluído um compromisso.

```
/*
 * método onMonthChanged
 * executado sempre que o usuário
 * altera o mês do calendário
 */
function onMonthChanged()
{
    // limpa a lista de tarefas
    $this->model->clear();
    // obtém o mês exibido atualmente
    $date = $this->calendar->get_date();
    $mes = $date[1] +1;

    // carrega todos os compromissos do mês
    // que for selecionado
    $result = $this->conn->Query("select *
from agenda where strftime('%m', data)
='{$mes}' order by data, hora");
    // percorre os compromissos, adicionando
no modelo de dados da lista
    foreach ($result as $row)
    {
        $dados = array($row['data'],
               $row['hora'],
               $row['titulo'],
               $row['descricao']);
        $this->model->append($dados);
    }
}
```

O próximo método é o onSave(). Este método é executado sempre que o usuário clicar no botão de salvar e será responsável por coletar os dados digitados a respeito do compromisso e por inserí-los no banco de dados, recarregando a listagem de tarefas logo após.

```
/*
 * método onSave
 * Salva um compromisso
 */
function onSave()
{
    // obtém os dados digitados
    $data       = $this->data->get_text();
    $hora       = $this->hora->get_text();
    $titulo     = $this->titulo->get_text();
    $buffer     = $this->descr->get_buffer();
    $descricao  = ...

    // insere no banco de dados
    $this->conn->Query("INSERT INTO agenda VA-
LUES ('$data', '$hora', '$titulo', '$descri-
cao')");

    // recarrega as tarefas do mês
    $this->onMonthChanged();
}
```

Quase terminando nossa Agenda, temos o mé-

todo onDelete(). Seu objetivo é obter o item selecionado na listagem de compromissos e perguntar ao usuário se ele deseja excluí-lo. Caso afirmativo, o programa irá excluir o item do banco de dados, baseado na data e hora do compromisso, recarregando a listagem de compromissos logo após.

```
/*
 * método onDelete
 * Exclui um compromisso
 */
function onDelete()
{
    // obtém o registro selecionado
    $selection =
        $this->list->get_selection();
    list ($model, $iter) =
        $selection->get_selected();
    // sel há seleção
    if ($iter)
    {
        // obtém a data e hora
        $data = $model->get_value($iter, 0);
        $hora = $model->get_value($iter, 1);

        // pergunta ao usuário
        $dialog = new GtkMessageDialog(...
           'Deseja excluir a tarefa ?');
        $resposta = $dialog->run();

        if ($resposta == Gtk::RESPONSE_YES)
        {
            // exclui registro
            $this->conn->Query("DELETE FROM
agenda WHERE data='$data' and hora='$hora'");
        }
        // fecha diálogo
        $dialog->destroy();
    }
    // recarrega as tarefas do mês
    $this->onMonthChanged();
}
```

Por último, temos o método que é executado quando a janela da aplicação é fechada, ou seja, destruída. Neste caso, estamos fechando a conexão PDO com o banco SQLite simplesmente destruindo o objeto de conexão.

```
/*
 * método __destruct
 * Executado quando janela for destruída
 */
function __destruct()
{
    // fecha conexão PDO
    unset($this->conn);
}

}
```

Agora que terminamos de escrever nossa classe, veja a seguir como ficou nosso programa. Nele, precisamos somente instanciar um objeto da classe Agenda. Como esta classe é filha de GtkWindow, utilizamos o método show_all() para exibi-la na tela.

```
$app = new Agenda;
$app->show_all();
Gtk::Main()
?>
```

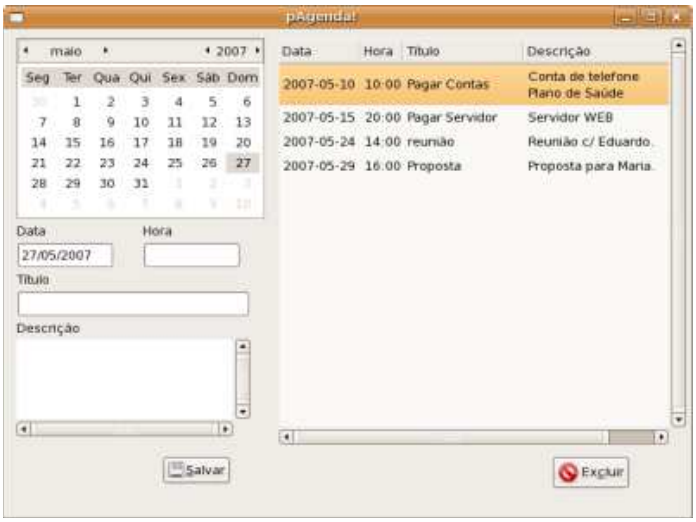

Figura 2 – Tela da Agenda riada

Considerações finais

Neste artigo escrevemos uma simples aplicação para controle de compromissos em PHP-GTK com menos de 200 linhas de código, descartando os comentários. Como o programa ficou um pouco grande para o formato da revista, optamos por cortar alguns pedaços, substituindo-os por "..." por motivos didáticos. Você pode fazer download da aplicação completa no site da Agenda, em http://pagenda.php-gtk.com.br

Referências e links sugeridos

[PHP-GTK Brasil] – http://www.php-gtk.com.br
[Livro PHP-GTK] – http://www.php-gtk.com.br/book
[Site do autor] – http://www.pablo.blog.br
[Site da Agenda] –http://pagenda.php-gtk.com.br

Pablo Dall'Oglio - pablo@dalloglio.net
Pablo Dall'Oglio é formado em Análise de Sistemas pela UNISINOS. Autor do livro sobre PHP-GTK pela novatec editora, programa em PHP-GTK desde sua criação em 2001. É membro do time de documentação e criador da comunidade brasileira de PHP-GTK (www.php-gtk.com.br). Atualmente, é diretor de tecnologia e proprietário da Adianti Solutions (www.adianti.com.br), onde atua como consultor de tecnologia e engenheiro de software. Pode ser contatado pelo e-mail pablo@php.net. Pablo Dall'Oglio é formado em Análise de Sistemas pela UNISINOS. Autor do livro sobre PHP-GTK pela novatec editora, programa em PHP-GTK desde sua criação em 2001. É membro do time de documentação e criador da comunidade brasileira de PHP-GTK (www.php-gtk.com.br). Atualmente, é diretor de tecnologia e proprietário da Adianti Solutions (www.adianti.com.br), onde atua como consultor de tecnologia e engenheiro de software. Pode ser contatado pelo e-mail pablo@php.net.