Como usar o SQLReactor para persistência de objetos PHP num banco de dados

Objetos PHP no banco

A persistência de objetos PHP em bancos de dados não requer operações complicadas. Basta um mapeador competente como o SQLReactor.

por Rafael Marques Martins

Uma ferramenta de ORM (*Object-Relational Mapping*, ou Mapeamento Objeto-Relacional) consiste em uma interface que implementa todos os métodos de acesso ao banco de dados, busca e alteração de registros, traduzindo-os para o conceito de objetos.

Listagem 1: Estrutura do código com o SQLReactor

```
01 <?php
02 include "SQLReactor/SQLReactor.php";
03
04 $connection = new SQLReactorConnection( "postgres://
➥sqlreactor:sqlreactor@localhost/exemplo_reactor" );
05 SQLReactor::setDefaultConnection( $connection );
06
07 //Aqui vai o mapeamento
08 //Aqui vão as operações em banco
09
10 $connection->close();
11 ?>
```

Utilizamos, portanto o conceito de objetos persistentes. Esses objetos serão armazenados em um banco de dados relacional, porém o conceito de banco de dados só existe no momento do mapeamento das classes em tabelas. Na prática, não mais se utiliza código SQL para inserir, alterar, excluir ou buscar registros no banco. Utilizamos o os métodos que a ferramenta de ORM disponibiliza para executar essas operações.

Portanto, utilizar uma ferramenta de ORM consiste basicamente em mapear e criar as estruturas do banco de dados e utilizar os métodos de busca e manipulação dos objetos. A ferramenta é res-

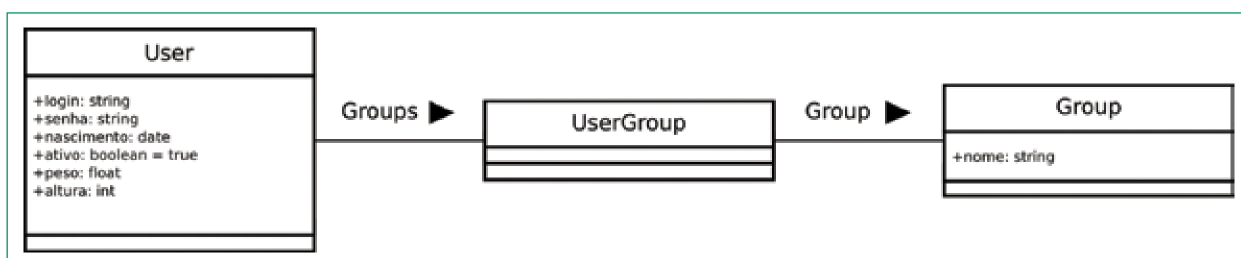

Figura 1 Diagrama de classes simples para exemplo.

ponsável por construir os comandos necessários para o sistema de banco de dados escolhido. Dessa forma, além de maior organização – pois o sistema fica completamente orientado a objetos –, obtém-se também maior liberdade quanto ao banco de dados a ser escolhido e uma maior facilidade para criar sistemas multibanco.

O SQLReactor

O SQLReactor é uma ferramenta de ORM de código aberto com suporte a MySQL, PostgreSQL, SQLite e Oracle. Funciona no PHP 5.0 ou superior e provê muitas opções de busca de dados e tradução automática de tipos de dados do PHP para o banco de dados e vice-versa.

Iniciando

Para iniciar o uso do SQLReactor, faça seu download em [1], descompacte os arquivos na estrutura do seu site e use a diretiva `include` para embutir o conteúdo do arquivo principal.

Depois disso, utilize a classe `SQLReactorConnection` para criar uma conexão com o banco de dados, defina-a como a conexão padrão, faça o mapeamento das classes e operações em banco e, no final, feche a conexão. A **listagem 1** mostra como fica a estrutura do código.

Nos exemplos deste artigo, todas as operações e mapeamentos são realizados em um único arquivo, para facilitar a leitura. Porém, a estrutura pode ser dividida pelo programador para uma maior organização do código.

Na **listagem 1**, utilizamos um banco de dados PostgreSQL. Para

Listagem 2: Exemplo de mapeamento

```
01 class User extends SQLReactor{
02   function __map(){
03     $this->login        = SQLReactor::StringCol( array( 'length' => 100, 'notNull' => true ) );
04     $this->password     = SQLReactor::StringCol( array( 'length' => 100, 'notNull' => true ) );
05     $this->birthday      = SQLReactor::DateCol();
06     $this->ativo     = SQLReactor::BoolCol( array( 'default' => true ) );
07     $this->weight        = SQLReactor::FloatCol();
08     $this->height        = SQLReactor::IntCol();
09
10     $this->unique( 'login' );
11
12     $this->groups       = SQLReactor::Backref( array( 'target' => array( 'UserGroup', 'user' ) ) );
13   }
14
15   function __setPassword( $value ){
16     return sha1( $value );
17   }
18 }
19
20 class Group extends SQLReactor{
21   function __map(){
22     $this->login        = SQLReactor::StringCol( array( 'length' => 100, 'notNull' => true ) );
23     $this->users         = SQLReactor::Backref( array( 'target' => array( 'UserGroup',
➥'group' ) ) );
24   }
25 }
26
27 class UserGroup extends SQLReactor{
28   function __map(){
29     $this->user          = SQLReactor::ForeignKey( array( 'target' => 'User' ) );
30     $this->group         = SQLReactor::ForeignKey( array( 'target' => 'Group' ) );
31
32     $this->primaryKey( 'userId', 'groupId' );
33   }
34 }
```

Listagem 3: Exemplo de criação das tabelas

```
01 <?php
02 include "open_connection.php";
03 include "mapping.php";
04
05 SQLReactor::createTable( 'User' );
06 SQLReactor::createTable( 'Group' );
07 SQLReactor::createTable( 'UserGroup' );
08
09 include "close_connection.php";
10 ?>
```

Listagem 4: Inserindo dados de exemplo

```
01 $user = new User();
02 $user->setLogin( 'user1' );
03 $user->setPassword( '123456' );
04 $user->setBirthday( mktime( 0, 0, 0, 3, 29, 1986 ) );
05 $user->save();
06 echo $user->id; //retorna o id do objeto inserido
07
08 $user = new User();
09 $user->login = 'user2';
10 $user->password = '123456';
11 $user->birthday = mktime( 0, 0, 0, 3, 29, 1989 );
12 $user->save();
13
14 $group = new Group();
15 $group->name = 'Group 1';
16 $group->save();
17
18 $group = new Group();
29 $group->name = 'Group 2';
20 $group->save();
21
22 $ug = new UserGroup();
23 $ug->userId = 1;
24 $ug->groupId = 1;
25 $ug->save();
26
27 $ug = new UserGroup();
28 $ug->userId = 1;
29 $ug->groupId = 2;
30 $ug->save();
31
32 $ug = new UserGroup();
33 $ug->userId = 2;
34 $ug->groupId = 2;
35 $ug->save();
```

usar outro banco de dados, será necessário mudar a URI passada para a classe `SQLReactorConnection`. Os formatos de URIs para outros bancos de dados encontram-se na **tabela 1**.

Mapeando

Este artigo mostra como fazer o mapeamento do diagrama de classes da **figura 1**, pois, apesar de simples, esse processo envolve grande parte dos conceitos necessários para o uso da ferramenta.

Para fazer o mapeamento no SQLReactor, é preciso definir as classes desejadas estendendo a classe `SQLReactor`. Todos os atributos devem ser mapeados dentro do método mágico `__map`. A classe também permite *setters* e *getters* mágicos usando o nome do atributo com os prefixos `__set` e `__get`, respectivamente (`__setPassword`, por exemplo).

Todas as classes automaticamente recebem um atributo `id` do tipo inteiro, que será a chave primária da tabela. A chave primária pode ser sobrescrita usando o método `$this->primaryKey` (veja a **listagem 2**).

O mapeamento completo do diagrama da **figura 1** encontra-se na **listagem 2**. Os possíveis tipos de atributos (isto é, colunas) do mapeamento estão descritos na **tabela 2**.

Na **listagem 2**, foi criado o setter mágico `__setPassword` para fazer com que a senha seja "criptografada" automaticamente quando o atributo `senha` for alterado.

Tabelas e objetos

Após fazer o mapeamento, basta utilizar os métodos do SQLReactor para criar as tabelas e manipular os objetos persistentes. A **listagem 3** contém um exemplo de código para a criação das tabelas.

Depois de criar as tabelas, já podemos começar a manipular objetos persistentes. Para criar um novo objeto, basta criar uma nova

instância da classe desejada e chamar o método `save` para persistir às alterações no banco. Após chamar o método `save` em um novo objeto, o SQLReactor automaticamente atualiza seu atributo `id` para coincidir com o id salvo no banco. A **listagem 4** contém um trecho de código para inserir dados de exemplo nas tabelas. O resultado dessas operações no banco de dados pode ser visto na **listagem 5**.

A partir deste ponto, basta utilizar os métodos de busca e manipulação de objetos providos pelo SQLReactor.

Os métodos disponíveis para busca de objetos incluem busca por id, busca usando filtro e listagem. Por exemplo, para buscar o objeto da classe `User` que contém id 1:

```
$user = new User( 1 );
```

A **listagem 6** realiza uma busca pelo objeto `User` de login igual a `user1` e com o atributo `ativo` marcado como `true`. O método `get` deve ser usado para retornar um único registro (ou uma instância vazia da classe passada, caso nada seja encontrado). Se mais de um objeto for encontrado com os parâmetros passados, é lançada uma exceção.

Também é possível utilizar o método `getList` para obter uma lista de objetos. Em todos os tipos de filtros, é possível navegar para os objetos ligados via `ForeignKey` ou `Backref` para filtrar o retorno. A **listagem 7** ilustra um exemplo disso, filtrando o resultado por `ativo`, pela data de nascimento maior ou igual a 01/01/1986 e pelo grupo. Neste trecho de código só serão retornados usuários que pertençam ao grupo cujo id seja igual a 2.

Em todos esses casos, o sistema recupera apenas os objetos do tipo `User`; mesmo que o filtro use outros objetos relacionados, eles não são trazidos do banco. Caso um atribu-

Listagem 5: Resultado da listagem 4 no banco de dados

```
exemplo_reactor=# select * from “user”;
id        | 1
login     | user1
password  | 7c4a8d09ca3762af61e59520943dc26494f8941b
birthday  | 1986-03-29
is_active | t
weight    |
height    |
----+---------------
id        | 2
login     | user2
password  | 7c4a8d09ca3762af61e59520943dc26494f8941b
birthday  | 1989-03-29
is_active | t
weight    |
height    |


exemplo_reactor=# select * from “group”;
id   | 1
name | Group 1
---+----
id   | 2
name | Group 2


exemplo_reactor=# select * from “user_group”;
user_id  | 1
group_id | 1
---+--
user_id  | 1
group_id | 2
---+--
user_id  | 2
group_id | 2
```

Tabela 1: Formatos de URIs para diferentes bancos de dados

Banco de dados	URI
PostgreSQL	`postgres://sqlreactor:sqlreactor@localhost/exemplo_reactor`
MySQL	`mysql://root:minhasenha@localhost/exemplo_reactor`
SQLite	`sqlite:///caminho/pro/arquivo.db`
Oracle	`oracle://sqlreactor:minhasenha@meuTNS`

Listagem 6: Busca de um objeto com filtros

```
01 $user = SQLReactor::get( 'User', array(
02   'filter' => array(
03     array( 'login', 'user1' ),
04     array( 'ativo', true ),
05   )
06 ) );
```

Listagem 7: Busca de uma lista de objetos

```
01 $list = SQLReactor::getList( 'User', array(
02     'filter' => array(
03         array( 'ativo', true ),
04         array( 'birthday', '>=', mktime( 0, 0, 0, 1, 1,
➥ 1986 ) ),
05         array( 'groups->group->id', 2 )
06     )
07 ) );
```

Listagem 8: Buscando objetos relacionados usando o eagerload

```
01 $list = SQLReactor::getList( 'User', array(
02     'filter' => array(
03         array( 'ativo', true ),
04         array( 'groups->group->id', 2 )
05     ),
06     'eagerload' => array( 'groups->group' )
07 ) );
```

Listagem 9: Exemplo completo de listagem

```
01 $list = SQLReactor::getList( 'User', array(
02   'filter' => array(
03     array( 'ativo', true ),
04     array( 'groups->group->id', 2 )
05   ),
06   'eagerload' => array( 'groups->group' ),
07   'limit' => 50,
08   'offset' => 0,
09   'orderBy' => 'birthday',
10   'direction' => 'asc'
11 ) );
```

Listagem 10: Obtendo o número de objetos no banco

```
01 $count = SQLReactor::count( 'User', array(
02     'filter' => array(
03         array( 'ativo', true ),
04         array( 'groups->group->id', 2 )
05     ) ) );
```

to do tipo `ForeignKey` ou `Backref` seja acessado, o sistema buscará os objetos relacionados automaticamente, porém fará uma nova consulta ao banco para isso.

Ao acessar `$user->groups` por exemplo, a ferramenta de ORM executará automaticamente uma consulta para buscar os objetos do tipo `UserGroup` que estão ligados ao usuário atual. Caso seja interesse do programador trazer os objetos `UserGroup` e `Group` (para exibir o nome dos grupos, por exemplo), usa-se uma técnica chamada *eager load*. Caso o parâmetro `eagerload` seja passado para uma busca, o SQLReactor automaticamente inclui os atributos ligados e traz os objetos no retorno. A **listagem 8** mostra como trazer as informações do grupo já na lista, utilizando apenas uma consulta ao banco de dados com o parâmetro `eagerload`.

Também é possível, nas listagens e contagens, passar o número máximo do registro a retornar e o índice do primeiro registro a ser retornado (conhecidos como `LIMIT` e `OFFSET` na maioria dos bancos de dados). As listagens também aceitam parâmetros de ordenação por um atributo e o método a ser usado na ordenação (ascendente ou descendente), definido no SQLReactor como `direction`.

A **listagem 9** mostra o uso desses parâmetros. Nela, utilizamos o mesmo filtro e `eagerload` definidos anteriormente, porém passando um número máximo de 50 objetos do tipo `User` (independentemente do número de grupos), iniciando no índice 0. Também solicita-se a ordenação dos objetos por data de nascimento em ordem crescente.

Também é possível utilizar um método de contagem de registros que não retorna os dados, apenas conta o número de objetos de um determinado tipo no banco, usando o mesmo método de filtragem dos métodos de busca. O código da **lis-**

Tabela 2: Tipos de atributos

Tipo	Descrição
`SQLReactor::IntCol`	Armazena números inteiros.
`SQLReactor::FloatCol`	Armazena números de ponto flutuante.
`SQLReactor::StringCol`	Armazena cadeias de caracteres. Se este campo receber o parâmtro `length`, o tipo de colunas do banco será `varchar(length)`; caso contrário será uma coluna de texto longo (normalmente Text ou CLOB, dependendo do banco de dados).
`SQLReactor::DateCol`	Armazena data.
`SQLReactor::DateTime`	Armazena data e hora.
`SQLReactor::TimeCol`	Armazena hora.
`SQLReactor::ForeignKey`	Define um relacionamento com outra classe. Recebe o parâmetro target no seguinte formato: `array( 'target' => 'NomeDaClasseAlvo' )` Este tipo cria automaticamente um atributo de mesmo nome, com o sufixo `Id` que contém o valor da chave estrangeira, enquanto o atributo original contém o objeto de `id` igual ao da chave estrangeira.
`SQLReactor::Backref`	Cria um atributo para navegar no sentido oposto ao da `ForeignKey`. Este relacionamento não vai para o banco de dados. É usado apenas para permitir a navegação entre os objetos. Recebe o parâmetro `target` no seguinte formato: `array( 'target' => array( 'ClasseQueContemAForeignKey', 'nomeDoAtributoForeignKey' ) )`

tagem 10 mostra como contar todos os usuários ativos que pertençam ao grupo de `id` igual a 2.

Cada execução do método `save` em um objeto já existente no banco de dados fará com que este objeto seja atualizado (caso algo nele tenha sido alterado). A **listagem 11** mostra como buscar um objeto por seu `id`, alterar o atributo ativo para `false` e persistir essa alteração no banco de dados.

Finalmente, objetos podem ser excluídos do banco de dados utilizando o método `delete` (**linha 6**).

Observações finais

Este artigo mostrou todos os passos para a utilização do SQLReactor em projetos ou sites orientados a objeto. Todas as operações foram descritas na forma de exemplo para facilitar a leitura, porém podem ser divididas em muitos arquivos, classes ou funções para maior organização.

As colunas to tipo `DateCol`, `DateTimeCol` e `TimeCol` aceitam entradas de data no formato de *unix timestamp* (padrão PHP) e cadeias de caracteres nos formatos `%y-%m-%d`, `%y-%m-%d %H:%M:%S` e `%H:%M:%S`, mas sempre retornam timestamps.

A utilização de uma ferramenta de ORM facilita muito o trabalho do programador – principalmente em sistemas orientados a objeto – e melhora significativamente a qualidade do código. Porém, o processo é mais pesado do que simplesmente obter os dados diretamente no banco (funções `mysql_fetch_array`, `pg_fetch_array` etc.).

Para maiores informações, pode-se consultar a página do projeto em [1] ou contactar o desenvolvedor da ferramenta (que é brasileiro) usando o endereço de email que consta nos cabeçalhos dos arquivos fonte. ■

Listagem 11: Persistindo alterações nos objetos

```
01 //Alterando objetos:
02 $user = new User( 1 );
03 $user->ativo = false;
04 $user->save();
05 //Apagando
06 $user->delete();
```

Mais informações

[1] Página do projeto SQLReactor: http://sourceforge.net/projects/sqlreactor

Sobre o autor

Rafael Marques Martins é analista programador graduado em Tecnologia de Sistemas de Informação pela Universidade Federal Fluminense.