# **ESTUDO DE CASO**

O foco desse trabalho é desenvolver um framework que atenda as necessidades dos programadores, com relação ao mapeamento de relacionamento entre tabelas dos bancos de dados e concilie com a facilidade de utilização.

## Tecnologias

Para o desenvolvimento do ORM a linguagem PHP foi escolhida. PHP é uma sigla em inglês que significa *Hypertext PreProcessor* (ou em português Pré-Processador de Hipertexto).

O PHP é uma linguagem de *script*, largamente utilizada, principalmente para desenvolvimento web. Seu uso é extremamente simples para iniciantes e oferece diversos recursos para desenvolvedores profissionais.

Outro motivo para o grande uso do PHP é o fato de ser *open source* e poder ser utilizado na maioria dos sistemas operacionais, incluindo Linux, variantes Unix, Mac OS e Microsoft Windows.

## Mapeamento

Para realizar o mapeamento dos modelos de dados, foi utilizado o conceito de *Annotations*.

As *annotations* são “etiquetas” que adicionam metadados relevantes sobre classes, métodos e propriedades. Ou seja, através do uso de *annotations*, pode-se adicionar às classes informações para mapear tabelas do banco de dados, e adicionar às propriedades da classe para mapear as colunas de uma tabela do banco de dados, para que posteriormente, em tempo de execução, os metadados indicados pelas *annotations* sejam analisados e a partir disso, o ORM irá trabalhar de acordo com essas informações.

Porém, no PHP, como em outras linguagens de programação como o Java, não existe uma estrutura nativa para adicionar esses metadados dentro do contexto da classe. Então, qual seria a solução para isso? Utilizar expressões regulares e a família de classes *Reflection* do PHP para resolver essa questão.

As liguagens de programação possuem uma estrutura para documentar as classes, propriedades e métodos, essa estrutura é o bloco de comentário. No PHP, essa estrutura é comumente conhecido como *PHPDoc*.

Para obter os comentários contidos nas classes e extrair as informações dos comentários, pode-se utilizar a família de classes *Reflection*, no caso, especificamente as classes *ReflectionClass* e *ReflectionProperty*.

Para os exemplos a seguir, considere a seguinte classe do modelos de dados:

1. <?php
2. namespace App;
3. /\*\*
4. \* @ORM/Entity
5. \* @ORM/Table(name=pessoa)
6. \*/
7. class Pessoa {
8. /\*\*
9. \* @ORM/Id
10. \* @ORM/Generated
11. \* @ORM/Column(name=pessoa\_id, type=int)
12. \*/
13. public $id;
14. /\*\*
15. \* @ORM/Column(name=nome, type=string, length=50)
16. \*/
17. public $nome;
18. /\*\*
19. \* @ORM/Column(name=data\_nasc, type=date)
20. \*/
21. public $dataNasc;
22. }

Listagem 1 – Classe Pessoa

Fonte: Autor, 2018

Extrair o *PHPDoc* de uma classe do modelo é realizado de maneira simples através do uso da classe *ReflectionClass*. Como no exemplo abaixo:

1. <?php
2. $reflection = new ReflectionClass(‘App\Pessoa’);
3. $doc = $reflection->getDocComment();
4. ?>

Listagem 2 – Exemplo de uso da classe ReflectionClass

Fonte: Autor, 2018

No exemplo mostrado na Listagem 2, na linha 2, é criado a instancia da classe *ReflectionClass*, passando como parâmetro para o construtor, o nome da classe do modelo. Já na linha 3, o método *getDocComment* retorna o bloco de comentário da classe informada no construtor.

Com a instância da classe *ReflectionClass*, é possível acessar as propriedades da classe do modelo. Por exemplo:

1. <?php
2. $reflection = new ReflectionClass(‘App\Pessoa’);
3. $doc = $reflection->getDocComment();
4. $properties = $reflection->getProperties();
5. foreach($properties as $property) {
6. $propertyDoc = $property->getDocComment();
7. }
8. ?>

Listagem 3 – Exemplo da classe ReflectionProperty

Fonte: Autor, 2018

Na linha 4, o método *getProperties* retorna um *array* contento instancias da classe *ReflectionProperity*, uma para cada propriedade contendo da classe. Então, para obter os blocos de comentário das propriedades da classe, percorre-se a o *array* na linha 6 e na linha 7 o método *getDocComment()* retorna o bloco de comentário da propriedade atual.

Agora, com os blocos de comentário em mãos, o próximo passo é utilizar a expressões regulares para extrair as *annotations* e suas informações. Expressão Regular é um recurso muito comum em diversas linguagens. Para executar uma expressão regular no PHP, usa-se duas das diversas funções nativas disponíveis chamadas “preg\_match” (PHP, 2018) e “preg\_match\_all” (PHP, 2018). Ambas executam uma expressão regular contra um texto procurando por referências, a diferença entre as duas é que, a função “preg\_match” para quando encontra a primeira correspondência e a função “preg\_match\_all” procura no texto inteiro e retorna todas as correspondências.

Na Listagem 1, a classe “Pessoa” possui algumas *annotations*, entre elas a *annotation* “@ORM/Table”, que informa, entre outras coisas, o noma da tabela em questão que deve ser mapeada. Então para identificar essa *annotation* e suas propriedades, para então extrair suas informações, é possível da seguinte maneira:

1. <?php
2. $reflection = new ReflectionClass(‘App\Pessoa’);
3. $doc = $reflection->getDocComment();
4. $findAnnotatoins = ‘/@ORM\/[@A-Za-z0-9=,\_\/\s\(\)\{\}]+/i’;
5. $findTable = ‘/Table\([@A-Za-z0-9=,\_\/\s\(\)]+\)/i’;
6. $findName = ‘/name[\s]?=[\s]?(\w+)/i’;
8. preg\_match\_all($findAnnotatoins, $doc, $annotations);
9. $annotations = join($annotations[0], ‘’);
10. preg\_match($findTable, $annotations, $table);
11. preg\_match($findName, $table[0], $name);
12. ?>

Listagem 4 – Exemplo de Expressão Regular

Fonte: Autor, 2018

As linhas 2 e 3 já foram mostradas anteriormente, mas nas linhas 4, 5 e 6, estão declaradas as expressões regulares que serão usadas para identificar a *annotation* e extrair as informações dela. Para um melhor entendimento das expressões regulares, ela pode ser quebrada em algumas partes menores:

* **“/”**

A primeira barra indica o começo da expressão regular e a última indica o começo da expressão regular;

* **“@ORM\/”**

Esse trecho indica que o texto correspondente precisa, obrigatóriamente, começar com o texto “@ORM/”

* **“[]”**

O trecho “[]” indica um conjunto, ou seja, a expressão deve considerar qualquer combinação dos caracteres entre colchetes como um texto válido.

* **“@A-Za-z0-9=,\_\/\(\)\{\}”**

O trecho acima indica que a expressão regular deve considerar o caracter “@”, todas letras maiúsculas e minúsculas, os números de 0 a 9, os caracteres “=”, “,”, “/”, “(“, “)”, “{“, “}”;

* **“\s”**

O trecho “\s” indica que a expressão regular deve considerar os caracteres de espaço e tabulação

* **“\w”**

O “\w” representa uma palavra com letras, números e underline

* **“i”**

A letra “i” sinaliza que a expressão regular deve ser *case insensitive*, ou seja, a expressão regular não deve diferenciar entre letras maiúsculas e minúsculas.

* **“+”**

O “+” indica que a expressão deve que o padrão que o precede ocorra uma ou mais vezes, por exemplo, “\w+”, o padrão indica que a expressão deve esperar uma ou mais palavras dentro do texto em que a expressão será aplicada.

* **“?”**

O “?” indica que a expressão deve que o padrão que o precede ocorra zero ou uma vez, por exemplo, “\w?”, o padrão indica que a expressão deve esperar zero ou uma (no caso a primeira) palavra dentro do texto em que a expressão será aplicada.

A combinação dessas partes e de várias outras possíveis, é o que cria o padrão a ser encontrado dentro do texto. As funções “preg\_match” e “preg\_match\_all” utilizam a expressão, que é informada no primeiro parâmetro da função, para encontrar as ocorrências dentro do bloco de texto, informado no segundo parâmetro e armazena as ocorrências em um *array* que é armazenado na variável informada no terceiro parâmetro.

Com todas as informações identificadas, elas serão armazenadas em um *array* acessível pelo ORM, para que posteriormente, sejam usadas para realizar os mapeamentos de consultas simples, consultas com relacionamentos (para identificar as colunas usadas para relacionar as duas tabelas), inclusões, alterações e exclusão na tabela a ser mapeada pela classe do modelo.

O leitura e extração das informações de classes para o mapeamento é realizado sob demanda, isso quer dizer que, o mapeamento de uma entidade irá ocorrer somente no momento me que o ORM precise trabalhar com a classe do modelo.

Por exemplo, considerando a classe “Pessoa” na Listagem 1, para realizar uma consulta à tabela “pessoa”, o ORM irá ler o mapeamento dessa classe somente no momento em que precisar montar a consulta pela primeira vez. Uma vez que o mapeamento esteja completo, essas informações estão disponíveis até o final do processamento do *script* PHP.

## Conexões

Para que se possa enviar comandos ao banco de dados e receber as repostas desses comandos, é necessário que exista uma conexão com o banco de dados para ser possível realizar essa comunicação.

As conexões que serão utilizadas pelo ORM devem ser declaradas em um arquivo com extensão “.php”. Por padrão, o arquivo é esperado que esteja na pasta raiz do ORM com o nome “connection.config.php”, ou seja, supondo que o ORM esteja localizado “/home/user/app/orm/”, então o caminho para o arquivo seria “/home/user/app/orm/connection.config.php”.

O arquivo deve conter um *array* com uma ou mais conexões, onde a chave da conexão é o nome identificador da conexão e o valor é um *array* contendo as informações da conexão. As informações variam de acordo com o banco de dados a ser utilizado. Na Listagem 5, pode-se ver um exemplo de arquivo de conexões para melhor compreensão.

1. <?php
2. return [
3. ‘exemplo-mysql’ => [
4. ‘db’ => ‘mysql’,
5. ‘version’ => ‘5.7.11’,
6. ‘host’ => ‘localhost’,
7. ‘schema’ => ‘app’,
8. ‘user’ => ‘root’
9. ‘pass’ => ‘root’
10. ],
11. ‘exemplo-sqlite’ => [
12. ‘db’ => ‘sqlite’,
13. ‘version’ => ‘3’,
14. ‘file’ => ‘../data/app-storage.sq3’,
15. ],
16. ];
17. ?>

Listagem 5 – Exemplo de arquivo de conexões

Fonte: Autor, 2018

De acordo com o exemplo da Listagem 5, duas conexões são definidas, na linha 3 está a declaração de uma conexão com o nome “exemplo-mysql” e na linha 11 a conexão com o nome “exemplo-sqlite”.

Os valores para a conexão “exemplo-mysql” é um *array* contendo as chaves “db”, “version”, “host”, “schema”, “user” e “pass”. A chave “db” contém o banco de dados a ser utilizado. A chave “version” indica a versão do banco de dados utilizado, no qual o driver deve corresponder à essa versão. A chave “host” é o endereço onde o banco de dados está localizado. A chave “schema” é o bando de dados (conjunto de tabelas) que será utilizado. As chaves “user” e “pass” são respectivamente o usuário e a senha de acesso ao banco de dados.

Os valores para a conexão “exemplo-sqlite” é um *array* contendo as chaves “db”, “version”, “file”. As chaves “db” e “version” funcionam da mesma maneira que a conexão anterior. A chave “file” indica o arquivo local o qual o banco de dados SQLite utilizará para armazenar os dados.

Para informar ao ORM qual (ou quais) conexão será utilizada na aplicação, deve ser feito através da classe principal do ORM conforme o exemplo a seguir:

1. <?php
2. $orm = ORM\*Orm*::getInstance();
3. $orm->setConnection(‘exemplo-mysql’);
4. ?>

Listagem 6 – Exemplo para utilizar a conexão

Fonte: Autor, 2018

A classe “ORM\Orm” concentra entre outras informações, as conexões com o banco de dados, e para isso ela utiliza o *design pattern* *Singleton*. O objetivo desse *design pattern* é definido no livro *“Design Patterns: Elements of Reusable Object-Oriented Software”* (1995) para garantir que existirá apenas uma instância de um determinado objeto e que essa instância estará disponível de forma púbica em todo o escopo da aplicação. Portanto, a classe “ORM/Orm” retém e centraliza todas as informações públicas pertinentes ao ORM, que estarão acessíveis para todo o ORM e no escopo da aplicação.

No entanto, cada banco de dados requer que uma conexão diferente seja estabelecida e como pode-se verificar na Listagem 5, cada conexão também requer informações diferentes. Para solucionar essa questão, é necessário delegar essa tarefa a alguém responsável por conhecer as particularidades de cada banco de dados. A solução para isso é o Driver.

### **Driver**

Criar a conexão com o banco de dados

Converter dados do PHP para o banco e do banco para o PHP

Contém as configurações para a conexão:

public $GENERATE\_ID\_TYPE;

public $GENERATE\_ID\_ATTR;

public $GENERATE\_ID\_QUERY;

public $SEQUENCE\_NAME = 'orm\_sequence';

public $IGNORE\_ID\_DATA\_TYPE = false;

public $FK\_ENABLE = true;

public $PAGE\_TEMPLATE;

public $TOP\_TEMPLATE;

public $DATA\_TYPES = [];

public $FORMATS = [ 'date' => 'Y-m-d', 'time' => 'H:i:s', 'datetime' => 'Y-m-d H:i:s' ];

### **Criação das tabelas**

As opções drop e create

Os callbacks beforeDrop e afterCreate

**Glossário**

Script:

Open Source:

Reflection:

Driver:

**Referências**

**PHP.** http://php.net/manual/en/preface.php. Última visita em 30 de abril de 2018.

**PHP.** http://php.net/manual/en/function.preg-match.php. Última visita em 30 de abril de 2018.

**PHP.** http://php.net/manual/en/function.preg-match-all.php. Última visita em 30 de abril de 2018.