1. **Introdução**

Lamberjack’s ORM é um *framework* ORM para a linguagem PHP. ORM é uma sigla em inglês que significa *Object-Relational Mapper*. Um ORM é uma ferramenta bastante útil no dia-a-dia do desenvolvedor de software.

O Lamberjack’s ORM trabalha com mapeamento de tabelas em classes do modelo de dados utilizando *Annotations*. O ORM permite ao desenvolvedor trabalhar com mais de uma conexão de banco de dados e criar as tabelas do banco de dados baseado nas classes do modelo.

1. **Instalação**

Para usar o Lamberjack’s ORM, pode-se obtê-lo no repositório **orm** no GitHub no link: https://github.com/dfrancklin/orm.

Basta então copiar a pasta “/orm” que foi baixada e para incluir o ORM no projeto, use o comando “require\_once”:

1. <?php
2. require\_once ‘./orm/load.php’;
3. ?>
4. **Definindo Conexões**

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 de conexões pode substituído da seguinte maneira:

1. <?php
2. $orm = ORM\*Orm*::getInstance();
3. $orm->setConnectionsFile(\_\_DIR\_\_.‘/db/connections.php’);
4. ?>

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.

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. ?>

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. $orm->addConnection(‘exemplo-sqlite’);
5. ?>

O método *setConnection* adiciona a conexão à lista de conexões que o ORM pode utilizar e faz com que a conexão informada seja a conexão padrão para o ORM, ou seja, qualquer operação que será realizada pelo ORM, se não for informada uma conexão explicitamente, o ORM irá assumir que a conexão que precisa ser usada é a conexão padrão.

Já o método *addConnection* apenas adiciona a conexão à lista de conexões que o ORM pode utilizar. A conexão padrão pode ser substituída a qualquer momento, para isso basta utilizar o método *setDefaultConnection*:

1. <?php
2. $orm = ORM\*Orm*::getInstance();
3. $orm->addConnection(‘exemplo-mysql’);
4. $orm->setDefaultConnection(‘exemplo-mysql’);
5. ?>
   1. **Criar Tabelas Automaticamente**

O ORM tem a habilidade de criar as tabelas a partir das classes modelo. Para que o ORM saiba como criar, é necessário informar o caminho para a pasta que contém os modelos e o *namespace* no momento em que estiver configurando a conexão no ORM. Por exemplo:

1. <?php
2. $orm = ORM\*Orm*::getInstance();
3. $orm->setConnection(‘exemplo-mysql’, [
4. ‘namespace’ => ‘App\Models’,
5. ‘modelsFolder’ => ‘/home/user/app/models’,
6. ‘create’ => true
7. ]);
8. ?>

Pode ser necessário também, apagar as tabelas antes de criá-las, para isso, basta informar também na configuração da conexão:

1. <?php
2. $orm = ORM\*Orm*::getInstance();
3. $orm->setConnection(‘exemplo-mysql’, [
4. ‘namespace’ => ‘App\Models’,
5. ‘modelsFolder’ => ‘/home/user/app/models’,
6. ‘create’ => true,
7. ‘drop’ => true
8. ]);
9. ?>

O ORM permite ainda, que uma ação seja executada antes de apagar as tabelas e uma ação após criar as tabelas. Essas ações podem ser úteis para criar uma rotina de *backup/restore* ou de migração de banco de dados. Para informar o ORM quais ações ele deve executar, basta fazer o seguinte:

1. <?php
2. $dbHelper = new App\Helpers\*InitDatabase()*;
3. $orm = ORM\*Orm*::getInstance();
4. $orm->setConnection(‘exemplo-mysql’, [
5. ‘namespace’ => ‘App’,
6. ‘modelsFolder’ => ‘/home/user/app/models’,
7. ‘drop’ => true,
8. ‘create’ => true,
9. ‘beforeDrop’ => [ $dbHelper, ‘beforeDrop’ ],
10. ‘afterCreate’ => [ $dbHelper, ‘afterCreate’ ]
11. ]);
12. ?>

Na linha 2, é criado uma instância da classe “App\Helpers\InitDatabase” e nas linhas 9 e 10, é informado para a conexão respectivamente, quais métodos devem ser executados antes de apagar as tabelas e depois de criá-las. Utilizando esses métodos é possível que o desenvolvedor desenvolva uma lógica de como realizar o backup das informações essenciais do banco de dados antes de apagar as tabelas e posteriormente restaurar essas informações após a criação das tabelas.

Os valores esperados pelas chaves “beforeDrop” e “afterCreate” podem ser também uma função anônima:

1. <?php
2. …
3. ‘beforeDrop’ => function($entityManager) { … },
4. ‘afterCreate’ => function($entityManager) { … }
5. …
6. ?>

Ou uma string contendo o nome de uma função:

1. <?php
2. …
3. ‘beforeDrop’ => ‘beforeDrop’,
4. ‘afterCreate’ => ‘afterCreate’
5. …
6. function beforeDrop($entityManager) { … }
7. function afterCreate($entityManager) { … }
8. ?>

O ORM passa por parâmetro uma instância de um *EntityManager* para os métodos ou funções que irão ser executados antes e depois do processo de criação das tabelas.

1. **Definindo Modelos**

* 1. **Relacionamentos**

1. **Entity Manager**
   1. **Transações**
      1. **Begin Transaction**

* + 1. **Commit**
    2. **Rollback**
  1. **Find**
  2. **List**
  3. **Query Builder**
     1. **Paginação**
     2. **Agregação**
     3. **Group by**
     4. **Order by**
     5. **Having by**
  4. **Save**
  5. **Remove**

1. **Logger**
2. **Driver**