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 uma instância de um *EntityManager* por parâmetro para os métodos ou funções que irão ser executados antes e depois do processo de criação das tabelas. Ele pode ser usado para realizar ações no banco de dados. O *EntityManager* será abordado mais a frente.

1. **Definindo Modelos**

Um modelo é uma classe que representa uma tabela no banco de dados e pode ser mapeada da classe para a tabela e da tabela para a classe em operações de consulta, inserção, alteração e deleção.

Para que um modelo possa representar devidamente uma tabela no banco de dados dentro do ORM, ela deve ser “anotada” utilizando o padrão de *annotation* definido pelo ORM.

* 1. ***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.

Abaixo, uma lista completa de as *annotations* e suas propriedades:

* + 1. ***Annotations* de Classes**
* ***Annotation*:** @ORM/Entity

**Descrição:** Define que a classe deve ser considerada como uma tabela no banco de dados.

**Preenchimento:** Obrigatório. O não preenchimento resulta em erro.

* ***Annotation*:** @ORM/Table

**Descrição:** Define informações sobre a tabela mapeada.

**Preenchimento:** Opcional.

**Propriedades:**

* + **Nome:** name.

**Descrição:** Define o nome da tabela mapeada.

**Preenchimento:** Opcional. Caso não preenchido, o ORM assume que o nome da tabela é o mesmo que o nome da classe.

* + **Nome:** schema

**Descrição:** Define qual é o conjunto de tabelas ou banco de dados ao qual a tabela mapeada em questão existe.

**Preenchimento:** Opcional. Caso não preenchido, o ORM assume que o schema a ser usado é o padrão definido na conexão ou nenhum, dependendo do banco de dados usado.

* + **Nome:** mutable

**Descrição:** Caso o valor seja “true” define que a tabela não pode ser modificada pelas operações de inserção, alteração e deleção. O valor padrão é “false”.

**Preenchimento:** Opcional. Assume o valor padrão caso não seja preenchido.

* + 1. ***Annotations* de Propriedades**
* ***Annotation*:** @ORM/Id

**Descrição:** Define que a propriedade representa a chave primária da tabela mapeada.

**Preenchimento:** Obrigatório. O não preenchimento resulta em erro.

* ***Annotation*:** @ORM/Generated

**Descrição:** Define que o valor da chave primaria é auto gerado, seja através de *sequence* ou qualquer tipo de *autoincrement* (isso é definido no driver para cada banco de dados).

**Preenchimento:** Opcional. O não preenchimento indica que o preenchimento e o incremento deverão ser feitos manualmente.

* ***Annotation*:** @ORM/Column

**Descrição:** Define informações sobre a coluna ser mapeada.

**Preenchimento:** Opcional. Assume os valores padrões das propriedades listadas a seguir.

**Propriedades:**

* + **Nome:** name.

**Descrição:** Define o nome da coluna a ser mapeada

**Preenchimento:** Opcional. Caso não seja preenchido, o ORM assume que o nome da coluna é o mesmo nome do atributo.

* + **Nome:** type.

**Descrição:** Define o tipo da coluna a ser mapeada

**Tipos:** string, int, float, lob (large object), date, time, datetime, bool

**Preenchimento:** Opcional. Caso não seja preenchido, o ORM assume que o tipo da coluna é string.

* + **Nome:** length.

**Descrição:** Define o tamanho da coluna a ser mapeada quando a coluna é do tipo “string”

**Preenchimento:** Opcional. Caso não seja preenchido, o ORM assume que o tamanho da coluna é 255.

* + **Nome:** scale.

**Descrição:** Define o tamanho da coluna a ser mapeada quando a coluna é do tipo “float”

**Preenchimento:** Opcional. Caso não seja preenchido, o ORM assume que o tamanho da coluna é 14.

* + **Nome:** precision.

**Descrição:** Define a precisão da coluna (quantidade de dígitos após a virgula) a ser mapeada quando a coluna é do tipo “float”

**Preenchimento:** Opcional. Caso não seja preenchido, o ORM assume que a precisão da coluna é 2.

* + **Nome:** unique.

**Descrição:** Se o valor do campo for “true”, define que o campo deve ser conter um valor único.

**Preenchimento:** Opcional. Valor padrão é “false”.

* + **Nome:** nullable.

**Descrição:** Se o valor do campo for “false”, define que o campo não pode receber valores nulos.

**Preenchimento:** Opcional. Valor padrão é “true”.

* + 1. ***Annotations* de Propriedades**
* ***Annotation*:** @ORM/HasOne

**Descrição:** Define um relacionamento de um para um. É necessário que a classe de referência tem um atributo equivalente à outra ponta do relacionamento com a *annotation* “@ORM/BelongsTo”.

**Preenchimento:** Opcional.

**Propriedades:**

* + **Nome:** class.

**Descrição:** Define qual classe deve ser referenciada no mapeamento.

**Preenchimento:** Obrigatório. O não preenchimento resulta em erro.

* + **Nome:** cascade.

**Descrição:** Define que as operações de inserção, alteração e deleção pode acontecer em cascata, ou seja, a operação realizada na classe que mapeia essa *annotation*, deve ser estendida para a classe referenciada.

**Valores:** INSERT, UPDATE, DELETE, ALL.

**Preenchimento:** Opcional. Caso não seja preenchida, a operação não é estendida.

* ***Annotation*:** @ORM/HasMany

**Descrição:** Define um relacionamento de um para muitos. É necessário que a classe de referência tem um atributo equivalente à outra ponta do relacionamento com a *annotation* “@ORM/BelongsTo”.

**Preenchimento:** Opcional.

**Propriedades:**

* + **Nome:** class.

**Descrição:** Define qual classe deve ser referenciada no mapeamento.

**Preenchimento:** Obrigatório. O não preenchimento resulta em erro.

* + **Nome:** cascade.

**Descrição:** Define que as operações de inserção, alteração e deleção pode acontecer em cascata, ou seja, a operação realizada na classe que mapeia essa *annotation*, deve ser estendida para a classe referenciada.

**Valores:** INSERT, UPDATE, DELETE, ALL.

**Preenchimento:** Opcional. Caso não seja preenchida, a operação não é estendida.

* ***Annotation*:** @ORM/BelongsTo

**Descrição:** Define a outra ponta dos relacionamentos um para um e um para muitos, ou seja, define a chave estrangeira para o relacionamento. É necessário que a classe de referência tem um atributo equivalente à outra ponta do relacionamento com a *annotation* “@ORM/HasOne” ou “@ORM/HasMany”.

**Preenchimento:** Caso uma relação um para um ou um para muitos seja definida, é obrigatório que a classe referenciada possua essa *annotation*.

**Propriedades:**

* + **Nome:** class.

**Descrição:** Define qual classe deve ser referenciada no mapeamento.

**Preenchimento:** Obrigatório. O Não preenchimento resulta em erro.

* + **Nome:** cascade.

**Descrição:** Define que as operações de inserção, alteração e deleção pode acontecer em cascata, ou seja, a operação realizada na classe que mapeia essa *annotation*, deve ser estendida para a classe referenciada.

**Valores:** INSERT, UPDATE, DELETE, ALL.

**Preenchimento:** Opcional. Caso não seja preenchida, a operação não é estendida.

* + **Nome:** optional.

**Descrição:** Define se o relacionamento é opcional, ou seja, indica que o valor pode ou não ser nulo.

**Preenchimento:** Opcional. Valor padrão é “false”

* ***Annotation*:** @ORM/JoinColumn

**Descrição:** Define as informações da coluna que deve ser a chave estrangeira. Somente a propriedade que possui a *annotation* “@ORM/BelongsTo” deve possuir essa *annotation* para complementar as informações.

**Preenchimento:** Opcional. Assume os valores padrões das propriedades listadas a seguir.

**Propriedades:**

* + **Nome:** name.

**Descrição:** Define o nome da coluna a ser mapeada como chave estrangeira.

**Preenchimento:** Opcional. O valor padrão é o nome da propriedade mais o sufixo “\_id”, por exemplo, “pessoa\_id”.

* ***Annotation*:** @ORM/ManyToMany

**Descrição:** Define um relacionamento de muitos para muitos. É necessário que a classe de referência tem um atributo equivalente à outra ponta do relacionamento com a *annotation* “@ORM/ManyToMany”.

**Preenchimento:** Caso uma relação muitos para muitos seja definida, é obrigatório que a classe referenciada possua essa *annotation*.

**Propriedades:**

* + **Nome:** class.

**Descrição:** Define qual classe deve ser referenciada no mapeamento.

**Preenchimento:** Obrigatório. O não preenchimento resulta em erro.

* + **Nome:** cascade.

**Descrição:** Define que as operações de inserção, alteração e deleção pode acontecer em cascata, ou seja, a operação realizada na classe que mapeia essa *annotation*, deve ser estendida para a classe referenciada.

**Valores:** INSERT, UPDATE, DELETE, ALL.

**Preenchimento:** Opcional. Caso não seja preenchida, a operação não é estendida.

* + **Nome:** mappedBy.

**Descrição:** Define que o lado principal do mapeamento é a classe referenciada e define também qual é o atributo ao qual é o equivalente. O lado principal pode definir também as informações da tabela de ligação.

**Preenchimento:** Opcional.

* ***Annotation*:** @ORM/JoinTable

**Descrição:** Define informações para a tabela de ligação. Somente o lado principal do relacionamento deve possuir essa *annotation* para complementar as informações.

**Preenchimento:** Opcional. Assume os valores padrões das propriedades listadas a seguir.

**Propriedades:**

* + **Nome:** tableName

**Descrição:** Define o nome da tabela de ligação.

**Preenchimento:** Opcional. O valor padrão e composto pelo nome das duas tabelas que compõe o relacionamento, por exemplo, “empregado\_role”.

* + **Nome:** schema

**Descrição:** Define qual é o conjunto de tabelas ou banco de dados ao qual a tabela de ligação em questão existe.

**Preenchimento:** Opcional. Caso não preenchido, o ORM assume que o schema a ser usado é o padrão definido na conexão ou nenhum, dependendo do banco de dados usado.

* + **Nome:** join

**Descrição:** Define o nome da coluna que é a chave estrangeira que aponta para o lado principal do relacionamento.

**Preenchimento:** Opcional. Assume os valores padrões das propriedades listadas a seguir.

**Propriedades:**

* + - **Nome:** name

**Descrição:** Define o nome da coluna a ser mapeada como chave estrangeira.

**Preenchimento:** Opcional. O valor padrão é o nome da propriedade mais o sufixo “\_id”, por exemplo, “empregado\_id”.

* + **Nome:** inverse

**Descrição:** Define o nome da coluna que é a chave estrangeira que aponta para o lado secundário do relacionamento.

**Preenchimento:** Opcional. Assume os valores padrões das propriedades listadas a seguir.

**Propriedades:**

* + - **Nome:** name

**Descrição:** Define o nome da coluna a ser mapeada como chave estrangeira.

**Preenchimento:** Opcional. O valor padrão é o nome da propriedade mais o sufixo “\_id”, por exemplo, “role\_id”.

* 1. **Exemplo de Mapeamento Simples**

Um exemplo básico de como criar uma classe do modelo, pode ser encontrado no exemplo a seguir. A classe “Empregado” mapeia a tabela “empregados” no banco de dados:

1. <?php
2. namespace App\Models;
3. /\*\*
4. \* @ORM/Entity
5. \* @ORM/Table(name=empregados)
6. \*/
7. class Empregado {
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. }
    1. **Relacionamentos**

Relacionamento entre tabelas é um recurso essencial nos banco de dados, para representar isso devidamente, no mundo orientado a objetos, existe as seguintes opções.

* + 1. **Um para Um**

Um exemplo básico de como mapear um relacionamento um para um, pode ser encontrado no exemplo a seguir. A classe “Empregado” mapeia o relacionamento através do atributo “$informacoes”:

1. <?php
2. namespace App\Models;
3. /\*\*
4. \* @ORM/Entity
5. \* @ORM/Table(name=empregados)
6. \*/
7. class Empregado {
8. ...
9. /\*\*
10. \* @ORM/HasOne(class=App\Models\EmpregadoInfo, cascade={ALL})
11. \*/
12. public $informacao;
13. ...
14. }

A classe “EmpregadoInfo” referenciada pela classe “Empregado” seria:

1. <?php
2. namespace App\Models;
3. /\*\*
4. \* @ORM/Entity
5. \* @ORM/Table(name=empregado\_info)
6. \*/
7. class EmpregadoInfo {
8. ...
9. /\*\*
10. \* @ORM/BelongsTo(class=App\Models\Empregado)
11. \* @ORM/JoinColumn(name=empregado\_id)
12. \*/
13. public $empregado;
14. ...
15. }
    * 1. **Um para Muitos**

Um exemplo básico de como mapear um relacionamento um para muitos, pode ser encontrado no exemplo a seguir. A classe “Cliente” mapeia o relacionamento através do atributo “$pedidos”:

1. <?php
2. namespace App\Models;
3. /\*\*
4. \* @ORM/Entity
5. \*/
6. class Cliente {
7. ...
8. /\*\*
9. \* @ORM/HasMany(class=App\Models\Pedido)
10. \*/
11. public $pedidos;
12. ...
13. }

A classe “Pedido” referenciada pela classe “Cliente” seria:

1. <?php
2. namespace App\Models;
3. /\*\*
4. \* @ORM/Entity
5. \*/
6. class Pedido {
7. ...
8. /\*\*
9. \* @ORM/BelongsTo(class=App\Models\Cliente)
10. \* @ORM/JoinColumn(name=cliente\_id)
11. \*/
12. public $cliente;
13. ...
14. }
    * 1. **Muitos para Muitos**

Um exemplo básico de como mapear um relacionamento muitos para muitos, pode ser encontrado no exemplo a seguir. A classe “Cliente” mapeia o relacionamento através do atributo “$pedidos”:

1. <?php
2. namespace App\Models;
3. /\*\*
4. \* @ORM/Entity
5. \*/
6. class Empregado {
7. ...
8. /\*\*
9. \* @ORM/ManyToMany(class=App\Models\Projeto)
10. \* @ORM/JoinTable(tableName=empregado\_projeto, join={ name=empregado\_id }, join={ name=projeto\_id })
11. \*/
12. public $projetos;
13. ...
14. }

A classe “Projeto” referenciada pela classe “Empregado” seria:

1. <?php
2. namespace App\Models;
3. /\*\*
4. \* @ORM/Entity
5. \*/
6. class Projeto {
7. ...
8. /\*\*
9. \* @ORM/ManyToMany(class=App\Models\Empregado, mappedBy=projetos)
10. \*/
11. public $empregados;
12. ...
13. }
14. **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**