RedBean

Un ORM pas comme les autres

Installation

```
composer require gabordemooij/redbean
```

voir la documentation

Utilisation

```php
require_once "../vendor/autoload.php";
//Importation de la bibliothèque
use RedBeanPHP\R as R;

//Initialisation de RedBean
R::setup();

//Création d'une entité
//Le nom de l'entité ne doit contenir
//que des caractères alphabétiques en minuscule
$book = R::dispense("book");

//Définition des données
$book->title = "Les chants de maldoror";
$book->author = "Lautréamont";

//Persistance de l'entité
$id = R::store($book);
```

Remarques

- La base de données est automatiquement générée (si aucune base n'est déclarée, RedBean utilisera une base SQLite)
- Le schéma des tables dépends de la saisie des données
- Le schéma est dynamique

Connexion

```php
//Fichier redbean-setup.php
//à intégrer au début de chaque page
require_once "../vendor/autoload.php";
use RedBeanPHP\R as R;

$dsn= "mysql:host=localhost;dbname=mydb;charset=utf8";
$user = "web";
$pass = "123";
//Initialisation de la connexion à une base MySQL ou MariaDB
R::setup($dsn, $user, $pass);
```

Fixtures

Chargement des données

```php
//Fichier book-fixtures.php

require_once "redbean-setup.php";

$bookData = [
    ["Les chants de Maldoror", "Lautréamont", 15.8, "Poésie", "Actes Sud"],
    ["Une saison en enfer", "Arthur Rimbaud", 11.5, "Poésie", "Gallimard"],
    ["Alcool", "Guillaume Apollinaire", 8.2, "Poésie", "Actes Sud"],
    ["Les chants de Maldoror", "Lautréamont", 15.8, "Poésie", "PUF"],
    ["Discours de la méthode", "René Descartes", 12.8, "Philosophie", "Hachette"],
    ["La République", "Platon", 11.8, "Philosophie", "Gallimard"],
    ["Pensées", "Blaise Pascal", 9.8, "Philosophie", "Hachette"],
    ["Le Banquet", "Platon", 12.8, "Philosophie", "PUF"],
];

//Suppression de tous les livres
R::wipe( "book" );

foreach ($bookData as $data){
    $book = R::dispense("book");

    $book->title = $data[0];
    $book->author = $data[1];
    $book->price = $data[2];
    $book->genre = $data[3];
    $book->editor = $data[4];

    R::store($book);
}
```

Le CRUD

Affichage de tous les livres

```php
$books = R::findAll("book");

foreach($books as $book){
    echo "<p> {$book->title} par {$book->author}</p>";
}
```

Recherche de livres

```
//Le deuxième argument utilise la syntaxe SQL
$books = R::find("book", "price >= 10");

foreach($books as $book){
    echo "<p> {$book->title} par {$book->author}</p>";
}
```

Multi critères

```php
$books = R::find("book", "price >= 10 and author = 'Lautréamont'");

foreach($books as $book){
    echo "<p> {$book->title} par {$book->author}</p>";
}
```

Charger un seul livre

Le deuxième argument correspond à la clef primaire, une propriété "id" automatiquement générée par RedBean

```php
$book = R::load("book", 1);


echo "<p> {$book->title} par {$book->author}</p>";
```

Modifier un livre

```php
//Chargement du livre
$book = R::load("book", 1);

//Modification du livre
$book->price = 5;

//Persistance
R::store($book);
```

Supprimer un livre

```php
//Chargement d'un livre
$book = R::load("book", 1);

//Suppression d'un livre
R::trash($book);
```

Supprimer des livres

```php
//Chargement de plusieurs livres
$books = R::find("book", "editor='Hachette'");

//autre possibilité en passant un tableau d'id
//à la méthode loadAll
$books = R::loadAll("book", [1,3,5]);

//supression des livres
R::trashAll($books);
```

Supprimer tout

```
//Supprime tous les livres
R::wipe("book");

//Supprime la base de données
R::nuke();
```

Les requêtes

Requête paramètrée

Même principe qu'avec PDO

```
$books = R::find(
    "book",// l'entité
    "author LIKE ?",//La requête avec marqueur
    ["%Rimbaud%"]//Les valeurs des marqueurs
);
```

Marqueurs nommés

Même principe qu'avec PDO

```
$books = R::find(
    "book",// l'entité
    "author LIKE :author",//La requête avec marqueur
    [":author"=>"%Rimbaud%"]//Les valeurs des marqueurs
);
```

Recherche d'un seul livre

```
$books = R::findOne(
    "book",
    "title = ?",
    ["La République"]
);
```

Prédicat IN

Rechercher plusieurs valeurs

```
$editorList = ["Hachette", "Grasset"];

//Génére une liste de marqueurs en fonction du tableau des critères
$placeHolders = R::genSlots($editorList);

$books = R::find(
    "book",
    "editor IN({$placeHolders})",
    $editorList
);
```

Order BY

```
$books = R::find(
    "book",
    "price > 9 ORDER BY author"
);
```

LIMIT

```
//Les 5 livres les plus chers
$books = R::findAll(
    "book",
    "ORDER BY price DESC LIMIT 5"
);
```

Find like

```
//Recherche les livres dont l'auteur
//est Platon ou Blaise Pascal
$books = R::findLike(
    "book",
    [
        "author" => ["Platon", "Blaise Pascal"]
    ]
);
```

Requête SQL

Pour aller au dela des limites de l'ORM

Exécution d'une requête

```
//Augmentation du prix de tous les livres
//bien plus efficace avec une requête SQL
R::exec("UPDATE book SET price=price*1.1");

//Même chose avec la suppression
R::Exec("DELETE FROM book WHERE editor=?", ["Hachette"]);
```

R::exec se comporte comme la commande execute de PDO

Requête SELECT

```
//Retourne un tableau ordinal de tableaux associatifs
$authors = R::getAll(
        "SELECT author, COUNT(id) as numberOfBooks
        FROM book GROUP BY author"
);

var_dump($authors);
```

Récupération des données

commande	Retour
`R::getAll()`	Une table
`R::getRow()`	Une seule ligne
`R::getCol()`	Une seule colonne
`R::getCell()`	Une seule valeur

Correspondance associative

R::getAssoc retourne un tableau indicé des lignes, cette méthode utilise comme clef la première colonne de la requête. Il est judicieux que cette première colonne possède une contrainte d'unicité.

```
var $books = R::getAssoc("SELECT id, title, author FROM book WHERE editor='PUF'");

/*
 array(2) {
  [73] =>
  array(2) {
    'title' =>
    string(22) "Les chants de Maldoror"
    'author' =>
    string(12) "Lautréamont"
  }
  [77] =>
  array(2) {
    'title' =>
    string(10) "Le Banquet"
    'author' =>
    string(6) "Platon"
  }
}
 */
```

Curseurs

Les méthodes find classiques chargent l'intégralité des données dans un tableau, si le nombre de résultats est important, il est possible d'utiliser un curseur qui chargera les données ligne par ligne.

```
$bookCollection = findCollection("book");


//Itération sur le curseur
while($book = $bookCollection->next()){
    echo $book->title;
}
```

Recherche création

Très utile lors de la création pour éviter de créer des doublons

```
$author = R::findOrCreate(
    "author",//nom de l'entité
    ["name"=>"Victor Hugo"]//données de l'entité
);
```

Recherche multiples

Permet de charger plusieurs entités avec une seule requête SQL

```php
$sql = "SELECT book.*, author.* FROM book INNER JOIN author ON author.id = book.author_id WHERE book.price >?";

$beans = R::findMulti(
    "book, author",//Liste des entités
    $sql,//Requête retournant les données
    [10]//critère
);

//La liste des livres
$books = $beans["book"];

//Le premier livre
$book = $books[0];
```

Les associations

Many to One

Plusieurs livres peuvent avoir le même auteur

```
//Création de l'entité author
$author = R::dispense("author");
$author->name = "Platon";

//Création de l'entité book
$book = R::dispense("book");
$book->title = "La République";
//La valeur de la clef autor est une référence à l'entité
$book->author = $author;
```

One to Many

Un auteur peut avoir écrit plusieurs livres

Attention : le nom de la variable qui stocke l'association doit commencer par own et continuer avec le nom de l'entité associée (ici book)

```php
//Création de l'auteur
$author = R::dispense("author");
$author->name = "Platon";

//Création des livres
list($book1, $book2) = R::dispense("book", 2);
$book1->title = "La République";
$book2->title = "Le Banquet";

//Association entre l'auteur et les livres
$author->ownBookList[] = $book1;
$author->ownBookList[] = $book2;


print_r(R::dump($author->ownBookList));
```

Association bidirectionnelle

```php
//Création de l'auteur
$author = R::dispense("author");
$author->name = "Platon";

//Création des livres
$book = R::dispense("book");
$book->title = "La République";
//Association entre les livres et l'auteur (many to one)
$book->author = $author;

//Association entre l'auteur et les livres (one to many)
$author->ownBookList[] = $book;
```

Many to Many

```php
//Création de plusieurs tags
$tags = R::dispense("tag",3);
$tags[0]->name = "Grece antique";
$tags[1]->name = "Philo";
$tags[2]->name = "Politique";

//Création de plusieurs livres
$books = R::dispense("book", 2);
$books[0]->title = "La République";
$books[1]->title = "Le Banquet";

//Associations des tags et des livres
//Le nom du tableau doit commencer par shared+nom de l'entité
$books[0]->sharedTagList[] = $tags[0];
$books[0]->sharedTagList[] = $tags[2];
$books[1]->sharedTagList[] = $tags[0];
$books[1]->sharedTagList[] = $tags[1];

R::storeAll($books);
R::storeAll($tags);
```

Utilisation des associations

```php
//Compter le nombre d'éléments
$nbTags = count($books[0].sharedTagList);

//Afficher tous les éléments
foreach ($books[0]->sharedTagList as $tag){
    echo $tag->name;
}
```

Association reflexive

```php
list($person, $friend) = R::dispense("person");

$person->name = "Joe";
$friend->name = "Jason";

//Association reflexive et bidirectionnelle
$person->sharedPersonList[] = $friend;
$friend->sharedPersonList[] = $person;
```

Filtrage SQL

Il est possible de rajouter des commandes SQL pour filtrer ou trier le résultat d'une association

```
//Liste de tous les livres associés à l'auteur triés par prix
$books = $author->with("ORDER BY price DESC")->ownBookList;

//Tous les livres associés à l'auteur dont le prix est inférieur à 10
$books = $author->withCondition("price < ?", [10])->ownBooList;

//Même chose mais avec un tri
$books = $author->withCondition("price < ? ORDER BY price asc", [10])->ownBooList;

//Avec une association sur la collection
$books = $author->withCondition("editor.name=?", ["Grasset"])->ownBookList;
```

Compte

```php
//Compter le nombre de livres
$numberOfBooks = R::count("book");

//Le nombre de livres de philo (si genre n'est pas une association)
$numberOfBooks = R::count("book", "genre=?", ["Philo"]);

//Le nombre de livres d'un auteur (avec association)
$numberOfBooks = $author->countOwn("book");

//Même chose avec association many to many
$numberOfTags = $book->countOwn("tag");
```

Compte avec filtrage

Le fitrage effectue une requête sans jointure, il faut donc récupérer l'identifiant de la clef étrangère.

```
//Affichage des requêtes SQL dans la console
R::fancyDebug(true);

//Liste les livres d'un auteur pour un éditeur particulier
$author = R::findOne("author", "name='Platon'");
$editor = R::findOne("editor", "name='PUF'");
$num = $author->withCondition("editor_id=?",[$editor->id])->countOwn("book");
```

Table d'association

Dans une association de plusieurs à plusieurs, si l'on souhaite ajouter de propriétés à l'association il faut définir une troisième entité

```php
//Les produits
$product = R::dispense("product");
$product->price = 5;
$product->name = "Clavier";

//La commande
$order = R::dispense("order");
$order->orderedAt = new DateTime();
$order->client = R::dispense("client");
$order->client->name = "Joe";

//La ligne de commande
$commandLine= R::dispense("commandline");
$commandLine->product = $product;
$commandLine->order = $order;
//La propriété qt ajoute une information à la double association
$commandLine->qt = 2;

//Association bi-directionnelles
$order->ownCommandline[] = $commandLine;
```

Quelques astuces

Importation

Hydratation d'une entité avec un tableau associatif

```
//les clefs du tableau correspondront aux propriétés de l'entité
$book = R::dispense("book");
$book->import(["title"=>"La peste", "price"=> 9]);

//Cela peut fonctionner également avec un super globale
$book->import($_POST);

//problème, $book aura une propriété submit

//solution 1, passer la liset des clefs à conserver en deuxième argument
$book->import($_POST,"title,price");

//solution 2 isoler les données du livres dans le formulaire
$book->import($_POST["book"]);
```

Importation depuis une entité

```php
$book = R::findOne("book","title='La peste'");

//Nouveau livre de Camus (récupère les infos du premier livre)
$newBook = R::dispense("book");
$newBook->import($book);
$newBook->title = "L'Étranger";
$newBook->price = 7;
```

Importation avec Dispense

On peut passer un tableau associatif à la méthode dispense à condition de préciser le type de l'entitité dans une clef _type

```
$data = [
    "_type" => "book",
    "title" => "Le Mythe de Sisyphe",
    "editor" => ["_type"=>"editor", "name"=>"Hachette"],
    "ownAuthor" => [
        ["_type" => "author", "name" => "Albert Camus"]
    ]
];

$book = R::dispense($data);
```

Table de référence

Une table qui ne fait que référencer un libéllé

```
//génère un tableau d'entités
$sexes = R::dispenseLabels("sex", ["femme","homme"]);

//Retourne un tableau indice de toutes les valeurs
$sexesArray = R::gatherLabels("sex");

//Utilisation
$person = R::dispense("person");
$person->name="Wakuko";
$person->sex = $sexes[0];
```

Énumération

Une liste de valeurs prédéterminées

```php
$persons = R::dispense("person", 3);

//Le deuxième appel à nationality:japonaise retournera un référence à l'entité
$persons[0]->name="Wakuko";
$persons[0]->nationality= R::enum("nationality:japonaise");
$persons[1]->name="Mary";
$persons[1]->nationality= R::enum("nationality:anglaise");
$persons[2]->name="Noriko";
$persons[2]->nationality= R::enum("nationality:japonaise");

R::storeAll($persons);

//Obtenir la liste des entités d'une énumeration
$allNationalities = R::enum("nationality");

//Requête sur une énumération
$japanesePersons = R::find("person",
    "nationality_id= ?",
    [R::enum("nationality:japonaise")->id]
);
```

Les tags

Catégoriser une entité avec une association de plusieurs à plusieurs

```
$posts = R::dispense("post", 2);

$posts[0]->title = "Les nouveautés de PHP7";
R::tag($posts[0], ["PHP", "web", "veille techno"]);

$posts[1]->title = "Angular 4 ce qui change";
R::tag($posts[1], ["Javascript", "angular", "web", "veille techno"]);

R::storeAll($posts);
```

Opérations sur les tags

```php
//Obtenir la liste des tags d'une entite
$tags = R::tag($posts[0]);

//Supprimer des tags
R::untag($posts[0], ["web"]);

//Obtenir la liste des entités taguées
$taggedPosts = R::tagged("post", ["PHP", "angular"]);

//Ajouter de nouveau tags
R::addTags($posts[1], ["front end", "typeScript"]);

//tester si une entité possède au moins un des tags de la liste
$test = R::hasTag($posts[1], ["web", "PHP"]);

//tester si une entité possède tous les tags de la liste
$test = R::tagged($posts[0], ["web", "PHP"] );
```

Déploiement

Avant de déployer l'application il faut geler le schéma

En effet, un schéma fluide est intéressant lors de la conception d'une application, mais suicidaire lors de son exploitation

```
R::freeze(true);
```

Conclusions

Inconvénients

- Fait bondir les DBA (et nombre d'examinateurs CDI)
- Pas d'auto-completion sur les propriétés des entités (une faute de frappe peut provoquer un bug difficile à repérer)
- Ne nous force pas à concevoir la structure de données avant de coder

Avantages

- Très bon outil de prototypage
- Fait gagner du temps
- Offre une première approche de l'ORM
- C'est juste fun