Синхронная ORM для MySql на Node.js

Данная ORM зависит от библиотеки mysql2 и является асинхронной, для работы с ней необходимо установить модуль mysql2, а также использовать конструкцию **async await**. ORM имеет инкапсулированную защиту от sql-injection.

Создание модели

Для подключения ORM нужно использовать следующее:

```
modelMysql = require('../../lib/Orm/mysqlOrm');
```

Создание модели осуществляется с помощью наследования ORM и добавления названия таблицы:

```
function TaskModel(){
    tableName = 'Task';
    modelMysql.call(this, tableName);
}
TaskModel.prototype = Object.create(modelMysql.prototype);
TaskModel.prototype.constructor = TaskModel;
```

Для того, чтобы можно было создавать объекты модели, нужно сделать экспорт:

```
module.exports = TaskModel;
```

Создание объекта модели

```
let Task = new TaskModel();
```

Select

Для select – запроса необходимо использовать метод find:

```
let tasks = await Task.find('all');
```

В переменной *tasks* в примере выше будет хранить массив обьектов языка javaScript, которые потом можно выводить в представлении с помощью любого удобного шаблонизатора.

Для того, чтобы получить одну запись можно использовать следующий вариант метода *find*:

```
let task = await Task.find('one', {where: 'id = 1'});
```

Такой вызов метода *find* вернет объект одной записи.

FindById

Для реализации частой задачи поиска записи по *id*, существует метод *findById*:

```
let task = await Task.findById(1);
```

Данный метод возвращает объект, содержащий одну запись из БД.

Where

Реализация условий выглядит следующим образом:

```
let data = await PostModel.find('all', {
    where: [
        ['id > ', 1],
    ],
});
```

Для нескольких условий можно использовать следующий вариант вызова метода find:

```
let data1 = await PostModel.find('all', {
    where: [
            ['id > ', 1, 'AND', '('],
            ['title = ', 'title', '', ')'],
});
```

Также вместо *and* можно использовать *or*, или комбинировать их.

Group

```
let task = await Task.find('all', {group: 'id'});
```

Для группировки в обратном порядке

```
let task = await Task.find('all', {group: 'id', groupDesc: true});
```

Order

```
let task = await Task.find('all', {order: 'id'});
```

Для группировки в обратном порядке

```
let task = await Task.find('all', {order: 'id', orderDesc: true});
```

Выбор полей

```
let task = await Task.find('all', {select: ['id, title, description']});
```

При таком вызове метода *find* получится запрос: '*SELECT id, title, description FROM `task`*'.

Having

```
let task = await Task.find('all', {select: ['id, SUM(count_result)'],
having: 'SUM(count_result)'});
```

Limit

```
let task = await Task.find('all', {limit: '5'});
let task = await Task.find('all', {limit: '5, 5'});
```

Remove

```
let task = await Task.remove({where: 'id = 1'}); //delete one record
let task = await Task.remove({where: '1'}); //delete all records
```

Данный метод возвращает true в случае успешного удаления и false в случае ошибки, это позволяет контролировать данный метод.

RemoveById

```
let task = await Task.removeById(1); //delete one record
```

Данный метод возвращает true в случае успешного удаления и false в случае ошибки, это позволяет контролировать данный метод.

Insert

Метод save позволяет сохранять новую запись.

```
let tasks = await Task.save({
        title: 'название',
        text: 'текст',
        description: 'описание',
        catalog_id: 1,
    });
```

text = 'текст', description = 'описание', catalog_id = '1'.

Update

Метод save также можно использовать для изменения записи если после объекта данных также указать id-записи.

```
let tasks = await Task.save({
        title: 'название',
        text: 'текст',
        description: 'описание',
        catalog_id: 1,
    }, '1');
```

Like

Реализация метода *like* выглядит следующим образом:

```
let data = await PostModel.find('all', {
      like: [
          ['title', '%o%', 'OR', '('],
          ['title', '%o%', 'AND', ')'],
          ['title', '%o%', '' ],
```

Метод *like* принимает в себя массив массивов, каждый массив первым элементом принимает поле, по которому будет осуществлен поиск, вторым ключ, по которому нужно делать поиск, третьим – параметр *AND* или *OR* (последнее условие в массиве принимает пустой параметр потому что для него данный параметр полностью игнорируется). Последний параметр может получить открывающуюся или закрывающуюся скобку, для создания более сложных условий (аналогично объекту *WHERE*). Для того, чтобы осуществить поиск по нескольким условиям, необходимо добавить несколько массивов в массив метода *like*.

Пример одного условия для Like

```
let data = await PostModel.find('all', {
    like: [
        ['title', '%o%'],
    ],
}
```

Join

Реализация метода *join* выглядит следующим образом:

```
let tasks = await Task.find('all', {
    join: [
        [ 'inner', 'catalog', 'task.catalog_id = catalog.id'],
    ]
};
```

Метод *join* принимает в себя массив массивов, каждый массив первым элементом принимает поле, в котором определяется тип join-запроса, вторым элементом принимает поле, которое хранит таблицу, по которой будет осуществлен join-запрос, а третьим элементом условие on. Для того, чтобы осуществить поиск по нескольким условиям, необходимо добавить несколько массивов в массив метода *join*.

Union

Для реализации вложенных запросов

```
let subQ = await Post.find('all', {select: ['1', 'text', 'image' ], sql: true});
let q = await Task.find('all', union: subQ);
```

В данном методе необходимо сделать дополнительный подзапрос, в котором свойство sql сделать true. Для подзапроса можно использовать метод query, а также написать запрос вручную.

Query

```
let task = await Task.query('Select * from User'); //delete one record
```

Метод для того, чтобы написать запрос полостью вручную.