Idiorm Documentation

Выпуск

Kirill SKV1991 Saltykov

17 November 2015

Оглавление

Здесь вы найдете полный перевод официальной документации.

Автор перевода: Кирилл skv1991 Салтыков

Содержание:

Глава 1

Философия

Закон Парето гласит, что *20% действий приносит 80% результата.* В терминах разработки ПО, это можно перевести приблизительно так: *20% сложностей дают 80% результата.* Другими словами, вы можете многого добиться, будучи не особо умным.

Idiorm умышленно создан простым. В то время, как другие ORM состоят из множества классов с сложной иерархий наследования, у Idiorm только один класс, `ORM`, который функционирует и как беглый API для создания запросов `SELECT` и как простой класс модели CRUD. Если моя догадка верна, этого должно быть достаточно для многих современных приложений. Посмотрим правде в глаза: большинство из нас не создает свой Facebook. Мы работает над проектами от маленьких до больших размеров, где акцент делается на простоту и быстрое развитие, а не на бесконечную гибкость и возможности.

Можно рассматривать **Idiorm** как *микро-ORM*. Он может быть как "галстук, идущий вместе со смокингом Slim" (заимствованный оборот из фразы в DocumentCloud). Или может быть одним из инструментов в генеральной уборке этих ужасающих, заваленных SQL-мусором наследников PHP приложений, которые вы должны поддерживать.

Idiorm также может предоставить хорошую базу, на которой строится более высокоуровневые, более сложные абстракции базы данных. Например, Paris это реализация паттерна Active Record pattern построенная на базе Idiorm.

Глава 2

Установка

2.1 Packagist

Данная библиотека доступна через Packagist с идентификаторами vendor и package: `j4mie/idiorm`

Пожалуйста, ознакомьтесь с Packagist documentation для более исчерпывающей информации.

2.2 Загрузка

Вы можете склонировать git-репозиторий, скачать idiorm.php или release tag и затем поместить файл idiorm.php в директорию vendors/3rd party/libs вашего проекта.

Глава 3

Конфигурация

Первое, что нужно знать об Idiorm - *вам не нужно объявлять какие-либо классы модели для её использования.* С почти любой другой ORM, первым шагом является установка моделей и их связка с таблицами из базы данных (через переменные конфигурации, XML файлы и тому подобное). С Idiorm, вы можете приступить к использованию ORM сразу же.

3.1 Установка

Первым делом, укажите файл исходного кода Idiorm в `require`:

```
<?php
require_once 'idiorm.php';
```

Затем передайте *Data Source Name* строку соединения в метод `configure` класса ORM. Он используется PDO для соединения с вашей базой данных. Для более детальной информации, смотри документацию PDO.

```
<?php
ORM::configure('sqlite:./example.db');
```

Вы так же можете передать имя пользователя и пароль в драйвер базы данных, используя параметры конфигурации `username` и `password`. Например, если вы используете MySQL:

```
<?php
ORM::configure('mysql:host=localhost;dbname=my_database');
ORM::configure('username', 'database_user');
ORM::configure('password', 'top_secret');
```

Смотрите так же секцию “Конфигурация” ниже.

3.2 Конфигурация

Помимо передачи DSN строки для подключения к базе данных (смотри выше), метод `configure` можно использовать и для установки некоторых других простых настроек ORM класса. Изменение настроек подразумевает передачу в метод `configure` пар ключ/значение, представляющих название параметра, который вы хотите поменять, и значение, которое хотите ему установить.

```
<?php
ORM::configure('название_параметра', 'значение_для_параметра');
```

Следующий метод используется для передачи множества пар ключ/значение за раз.

```
<?php
ORM::configure(array(
    'название_параметра_1' => 'значение_для_параметра_1',
    'название_параметра_2' => 'значение_для_параметра_2',
    'etc' => 'etc'
));
```

Для чтения текущей конфигурации, используйте метод `get_config`.

```
<?php
$isLoggingEnabled = ORM::get_config('logging');
ORM::configure('logging', false);
// какой-то бешенный цикл, который мы не хотим добавлять в лог
ORM::configure('logging', $isLoggingEnabled);
```

3.2.1 Подробности аутентификации в базе данных

Параметры: `username` и `password`

Некоторые адаптеры баз данных (такие как MySQL) требуют раздельной передачи имени пользователя (username) и пароля (password) в DSN строке. Эти параметры позволяют вам передать их. Типичная настройка соединения MySQL может выглядеть так:

```
<?php
ORM::configure('mysql:host=localhost;dbname=my_database');
ORM::configure('username', 'database_user');
ORM::configure('password', 'top_secret');
```

Или вы можете соединить настройку соединения в одну строку, используя массив конфигурации:

```
<?php
ORM::configure(array(
    'connection_string' => 'mysql:host=localhost;dbname=my_database',
    'username' => 'database_user',
    'password' => 'top_secret'
));
```

3.2.2 Наборы результатов

Параметр: `return_result_sets`

Коллекция результатов данных может быть возвращена в качестве массива (по-умолчанию) или в виде результирующего набора. Смотрите документацию о *find_result_set()* для более подробной информации.

```
<?php
ORM::configure('return_result_sets', true); // возвращает результирующий набор
```

3.2.3 PDO Параметры драйвера

Параметр: `driver_options`

Некоторые адаптеры базы данных требуют (или позволяют использовать) массив параметров конфигурации для конкретного драйвера. Это позволяет вам передавать эти параметры через конструктор

PDO. Для более подробной информации, смотрите документацию PDO. Например, чтобы заставить драйвер MySQL использовать кодировку UTF-8 для соединения:

```
<?php
ORM::configure('driver_options', array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES utf8'));
```

3.2.4 PDO Режим ошибок

Параметр: `error_mode`

Можно использовать для установки параметра `PDO::ATTR_ERRMODE` у класса соединения с базой данных, используемой Idiorm. Должна быть передана одна из объявленных в классе констант PDO. Пример:

```
<?php
ORM::configure('error_mode', PDO::ERRMODE_WARNING);
```

Параметром по-умолчанию является `PDO::ERRMODE_EXCEPTION`. Для более подробной информации о доступных режимах ошибок, смотри документация PDO - присвоение атрибута.

3.2.5 PDO доступ к объекту

Если он когда-то потребуется, то объект PDO, используемый в Idiorm может быть получен напрямую через `ORM::get_db()`, или установлен напрямую через `ORM::set_db()`. Однако это редкое явление.

После того, как заявление было выполнено с помощью любых средств, таких, как `::save()` или `::raw_execute()`, используемый экземпляр `PDOStatement` может быть получен через `ORM::get_last_statement()`. Это может быть полезно для доступа к `PDOStatement::errorCode()`, если исключения в PDO отключены, или для доступа к методу `PDOStatement::rowCount()`, который возвращает различные результаты, относительно основной базы данных. Чтобы узнать больше, смотри PDOStatement документация.

3.2.6 Идентификатор символа кавычек

Параметр: `identifier_quote_character`

Устанавливает символ, используемый для идентификации кавычек (например имени таблицы, имени столбца). Если не задан, то будет определен автоматически в зависимости от драйвера базы данных, используемого в PDO.

3.2.7 ID столбца

По-умолчанию, ORM предполагает, что у всех ваших таблиц есть колонка первичного ключа(primary с именем `id`. Есть два способа это переопределить: для всех таблиц в базе данных, или на основе каждой таблицы.

Параметр: `id_column`

Этот параметр используется для конфигурации имени столбца первичного ключа (primary key) всех таблиц. Если ваша ID-колонка имеет название `primary_key`, используйте:

```
<?php
ORM::configure('id_column', 'primary_key');
```

Вы можете указать составной первичный ключ, используя массив:

```
<?php
ORM::configure('id_column', array('pk_1', 'pk_2'));
```

Примечание: Если вы используете столбец с auto-increment в составном первичном ключе, то он должен быть объявлен первым в массиве.

Параметр: `id_column_overrides`

Этот параметр используется для указания имени столбца первичного ключа для каждой таблицы отдельно. Он принимает ассоциативный массив, описывающий название таблиц и имен столбцов. Например, если, имена ID-столбца включают имя таблицы, то вы можете использовать следующую настройку:

```
<?php
ORM::configure('id_column_overrides', array(
    'person' => 'person_id',
    'role' => 'role_id',
));
```

Как и параметр `id_column`, вы можеите указать составной первичный ключ, используя массив.

3.2.8 Limit clause style

Параметр: `limit_clause_style`

Вы можете установить стиль ограничения в конфигурации. Это делается для облегчения ограничений в стиле MS SQL, использующий синтаксис `TOP`.

Допустимыми значениями являются `ORM::LIMIT_STYLE_TOP_N` и `ORM::LIMIT_STYLE_LIMIT`.

3.2.9 Лог запросов

Параметр: `logging`

Idiorm может записывать в лог все вызываемые запросы. Для включения лога запросов, установите параметр `logging` на `true` (по-умолчанию он имеет значение `false`).

Когда лог запросов включен, вы можете использовать два статических метода для доступа к логу. `ORM::get_last_query()` возвращает самый последний вызванный запрос. `ORM::get_query_log()` возвращает массив всех вызванных запросов.

3.2.10 Logger запросов

Параметр: `logger`

Можно передать в этот параметр конфигурации `callable`, который будет вызываться для каждого запроса, вызываемого idiorm. В PHP `callable` зовется все, что может быть вызвано, как если бы это была функция. Чаще всего это будет представляться в виде анонимной функции.

Это полезный параметр, если вы хотите отслеживать лог запросов с помощью внешней библиотеки, так как он позволяет получить все то, что происходит внутри функций.

```
<?php
ORM::configure('logger', function($log_string, $query_time) {
    echo $log_string . ' in ' . $query_time;
});
```

3.2.11 Кэширование запросов

Параметр: `caching`

Idiorm может кэшировать выполняемые запросы во время обращения. Для включения кэширования запросов, установите параметру `caching` значение `true` (по-умолчанию имеет значение `false`).

```
<?php
ORM::configure('caching', true);
```

Параметр: `caching_auto_clear`

По-умолчанию, кэш Idiorm никогда не очищается. Если вы хотите, чтобы он очищался после сохранения, установите параметр `caching_auto_clear` на значение `true`

```
<?php
ORM::configure('caching_auto_clear', true);
```

Когда включено кэширование запросов, Idiorm будет кэшировать результаты каждой выборки `SELECT`, которая была вызвана. Если Idiorm сталкивается с запросом, который уже был вызван, то он извлечет результаты прямо из кэша и не будет обращаться к базе данных.

Предупреждения и подводные камни

- Обратите внимание что для кэширования используется внутренняя память, сохраняющаяя данные в пределах одного запроса. Это *не* замена для существующих систем кэширования вроде Memcached.
- Кэш Idiorm построен очень просто, и не пытается стать недействительным, если данные изменились. Это означает, если вы выполните запрос для извлечения каких-то данных, измените их и сохраните, а далее вызовите такой же запрос снова, то результаты будут устаревшими (т.е., они не будут отражать изменений). Это может привести к потенциальным трудоуловимым ошибкам в вашем приложении. Если у вас включено кэширование и вы заметили странное поведение, отключите его и попробуйте снова. Если вам все-таки нужно выполнять такие операции, но вы хотите оставь кэш включенным, то можно вызвать метод `ORM::clear_cache()` для очистки всех кэшированных запросов.
- Включение кэширования увеличивает расход памяти для приложения, так как все строки базы данных, полученные по время каждого запроса будут храниться в памяти. Если вы работаете с большими объемами данных, то лучше будет отключить кэш.

Произвольное кэширование

Если вы хотите использовать произвольные функции кэширования, вы можете задать их в параметрах конфигурации.

```
<?php
$my_cache = array();
ORM::configure('cache_query_result', function ($cache_key, $value, $table_name, $connection_name) use (&$my_cache)
    $my_cache[$cache_key] = $value;
});
ORM::configure('check_query_cache', function ($cache_key, $table_name, $connection_name) use (&$my_cache) {
    if(isset($my_cache[$cache_key])){
       return $my_cache[$cache_key];
    } else {
    return false;
    }
```

```
});
ORM::configure('clear_cache', function ($table_name, $connection_name) use (&$my_cache) {
     $my_cache = array();
});

ORM::configure('create_cache_key', function ($query, $parameters, $table_name, $connection_name) {
    $parameter_string = join(',', $parameters);
    $key = $query . ':' . $parameter_string;
    $my_key = 'my-prefix'.crc32($key);
    return $my_key;
});
```

Глава 4

Запросы

Idiorm предоставляет fluent interface позволяющий построение простых запросов без единого использования SQL. Если вы использовали jQuery вообще, то будете уже знакомы с концепцией fluent interface. Просто напросто это значит что вы можете *сцеплять в цепочку* вызов методов вместе, один после другого. Это может сделать ваш код более читабельным, нанизывая методы друг на друга, как-будто вы составляете предложение.

Все запросы в Idiorm начинаются с вызова статического метода `for_table` класса ORM. Это сообщает ORM какую таблицу использовать при построении запроса.

*Обратите внимание, что этот метод **не* экранирует свои параметры запроса, так что имя таблицы **не** должно быть передано напрямую от от вводимых пользователем данных.*

Вызовы метода добавляют фильтры и ограничения в ваш запрос, нанизываясь друг на друга. Наконец, цепочка заканчивается вызовом метода `find_one()` или `find_many()`, которые выполняют запрос и возращают результат.

Начнем с простых примеров. Скажим у нас есть таблица `person` содержащая столбцы `id` (первичный ключ записи - Idiorm предполагает что столбец первчиных ключей называется `id` но это можно настроить, смотри ниже), `name`, `age` и `gender`.

4.1 Примечание по PSR-1 и стилю camelCase

Все методы, описанные в документации могут так же быть вызваны, используя стандарт PSR-1: подчеркивания (_) заменяются на стиль написания camelCase. Далее приведен пример одной цепочки запроса, конвертированной в совместимый с PSR-1 стиль.

```php
<?php
// документация и стиль по-умолчанию
$person = ORM::for_table('person')->where('name', 'Fred Bloggs')->find_one();

// PSR-1 совместимый стиль
$person = ORM::forTable('person')->where('name', 'Fred Bloggs')->findOne();
```

Как вы можете заметить, любой метод может быть изменен из формата с подчеркиваниями (_) как в документации в стиль camelCase.

4.2 Одиночные записи

Любая цепочка методов, заканчивающаяся на `find_one()` вернет либо *единичный* экземпляр класса ORM представляющий ряд из базы данных по указанному запросу, или `false` если не было найдено удовлетворяющих запросу записей.

Чтобы найти одиночную запись, где столбец `name` имеет значение "Fred Bloggs":

```
<?php
$person = ORM::for_table('person')->where('name', 'Fred Bloggs')->find_one();
```

Грубо переводя на язык SQL, это: `SELECT * FROM person WHERE name = "Fred Bloggs"`

Чтобы найти единичную запись по ID, вы можете передать ID прямо в метод `find_one`:

```
<?php
$person = ORM::for_table('person')->find_one(5);
```

Если вы используете составной первичный ключ, то можете найти записи используя массив в качестве параметра:

```
<?php
$person = ORM::for_table('user_role')->find_one(array(
    'user_id' => 34,
    'role_id' => 10
));
```

4.3 Множество записей

Любая цепочка методов, заканчивающаяся на `find_many()` вернет *массив* экземпляров ORM-класса, по одному для каждой удовлетворяющей запросу строки. Если не было найдено ни одной строки, то будет возвращен пустой массив.

Чтобы найти все записи в таблице:

```
<?php
$people = ORM::for_table('person')->find_many();
```

Чтобы найти все записи, где `gender` равен `female`:

```
<?php
$females = ORM::for_table('person')->where('gender', 'female')->find_many();
```

4.3.1 Как результирующий набор

Вы так же можете найти множество записей в качестве результирующих наборов вместо массива экземплятров Idiorm. Это дает преимущество в том, что вы можете запустить пакетные операции на наборе результатов.

Итак, для примера, вместо этого:

```
<?php
$people = ORM::for_table('person')->find_many();
foreach ($people as $person) {
    $person->age = 50;
```

```
    $person->save();
}
```

Вы можете использовать это:

```
<?php
ORM::for_table('person')->find_result_set()
->set('age', 50)
->save();
```

Чтобы это сделать, замените любой вызов метода `find_many()` методом `find_result_set()`.

Результирующий набор ведет себя так же, как и массив, так что вы можете использовать на нем *count()* и *foreach* как и с массивом.

```
<?php
foreach(ORM::for_table('person')->find_result_set() as $record) {
    echo $record->name;
}
```

```
<?php
echo count(ORM::for_table('person')->find_result_set());
```

4.3.2 Как ассоциативный массив

Так же вы можете найти множество записей в виде ассоциативного массива, вместо экземпляров Idiorm. Для этого замените любой вызов метода `find_many()` на метод `find_array()`.

```
<?php
$females = ORM::for_table('person')->where('gender', 'female')->find_array();
```

Это полезно, если вам нужно преобразовать результат запроса в последовательную форму записи(сериализация массива) для JSON, и вам не нужно дополнительной возможности обновлять возвращаемые данные.

4.4 Подсчет результатов

Для подсчета числа строк, возвращаемых запросом, вызовите метод `count()`.

```
<?php
$number_of_people = ORM::for_table('person')->count();
```

4.5 Фильтрация результатов

Idiorm предоставляет семейство методов, позволяющих извлечь только те записи, которые удовлетворяют определенному условию(ям). Эти методы можно вызывать множество раз для построения запроса, а fluent interface у Idiorm позволяет строить *цепочку* из таких методов, для построения читабельных и простых к пониманию запросов.

4.5.1 *Предостережения*

Только подмножество доступных условий, поддерживаемых SQL доступны при использовании Idiorm. Кроме того, все пункты `WHERE` будут соединены с использованием `AND` при выполнении запроса. Поддержка `OR` в пунктах `WHERE` в настоящее время отстутствует.

Данные ограничения являются преднамеренными: ведь это наиболее используемые критерии, и избегая поддержки очень сложных запросов, код Idiorm может оставаться маленьким и простым.

Некоторая поддержка более сложных условий и запросов реализована в методах `where_raw` и `raw_query` (смотрите ниже). Если вы поймете, что чаще нуждаетесь в в большем функционале, нежели содержит Idiorm, то возможно пришло время рассмотреть более полнофункциональный ORM.

4.5.2 Равенство: `where`, `where_equal`, `where_not_equal`

По-умолчанию, вызывая `where` с двумя параметрами (название столбца и значение), они будут соединены, используя оператор равенства (=). Например, вызов `where('name', 'Fred')` вернет следующее: `WHERE name = "Fred"`.

Если ваш стиль написания кода направлен на ясность написанного, а не на краткость, то можно использовать метод `where_equal` идентичный методу `where`.

Метод `where_not_equal` добавляет пункт `WHERE column != "value"` к вашему запросу.

Можно указать множество столбцов и их значений в пределах одного вызова. В этом случае, вам нужно передать ассоциативный массив в качестве первого параметра. В нотации массива, ключи используются как названия стобцов.

```
<?php
$people = ORM::for_table('person')
            ->where(array(
                'name' => 'Fred',
                'age' => 20
            ))
            ->find_many();

// Создаст следующий запрос SQL:
SELECT * FROM `person` WHERE `name` = "Fred" AND `age` = "20";
```

4.5.3 Короткая запись: `where_id_is`

Это простой вспомогательный метод, для составления запроса по первичному ключу таблицы. Смотрит относительно ID столбца, указанного в конфигурации. Если вы используете составной первичный ключ, то нужно передать массив, где ключом является название столбца. Столбцы, не принадлежащие к этому ключу будут игнорироваться.

4.5.4 Короткая запись: `where_id_in`

Этот вспомагательный метод аналогичен `where_id_is`, но он ожидает массив первичных ключей для выборки. Так же понимает и составной первичный ключ.

4.5.5 Меньше чем / больше чем: `where_lt`, `where_gt`, `where_lte`, `where_gte`

Есть четыре метода, доступные для неравенств:

- Меньше чем (less than): `$people = ORM::for_table('person')->where_lt('age', 10)->find_many();`
- Больше чем (greater than): `$people = ORM::for_table('person')->where_gt('age', 5)->find_many();`
- Меньше или равен (less than or equal): `$people = ORM::for_table('person')->where_lte('age', 10)->find_many();`
- Больше или равен (greater than or equal_: `$people = ORM::for_table('person')->where_gte('age', 5)->find_many();`

4.5.6 Сравнение строк: where_like и where_not_like

Для добавления пункта `WHERE ... LIKE`, используйте:

```
<?php
$people = ORM::for_table('person')->where_like('name', '%fred%')->find_many();
```

Аналогично и для `WHERE ... NOT LIKE`, используйте:

```
<?php
$people = ORM::for_table('person')->where_not_like('name', '%bob%')->find_many();
```

4.5.7 Множественные условия OR

Можно добавить простое условие OR в тот же пункт WHERE используя `where_any_is`. Если вам нужно указать множество условий, используйте массив элементов. Каждый элемент будет ассоциативным массивом, содержащим множество условий.

```
<?php
$people = ORM::for_table('person')
            ->where_any_is(array(
                array('name' => 'Joe', 'age' => 10),
                array('name' => 'Fred', 'age' => 20)))
            ->find_many();

// Создаст SQL запрос:
SELECT * FROM `widget` WHERE (( `name` = 'Joe' AND `age` = '10' ) OR ( `name` = 'Fred' AND `age` = '20' ));
```

По-умолчанию, оператор равенства используется для каждого столбца, но его можно переопределить для любого столбца, используя второй параметр:

```
<?php
$people = ORM::for_table('person')
            ->where_any_is(array(
                array('name' => 'Joe', 'age' => 10),
                array('name' => 'Fred', 'age' => 20)), array('age' => '>'))
            ->find_many();

// Создаст SQL запрос:
SELECT * FROM `widget` WHERE (( `name` = 'Joe' AND `age` > '10' ) OR ( `name` = 'Fred' AND `age` > '20' ));
```

Если вы хотите задать свой оператор по-умолчанию для всех столбцов, то нужно передать его как второй параметр:

```php
<?php
$people = ORM::for_table('person')
            ->where_any_is(array(
                array('score' => '5', 'age' => 10),
                array('score' => '15', 'age' => 20)), '>')
            ->find_many();

// Создаст SQL запрос:
SELECT * FROM `widget` WHERE (( `score` > '5' AND `age` > '10' ) OR ( `score` > '15' AND `age` > '20' ));
```

4.5.8 Определение принадлежности: where_in и where_not_in

Для добавления пунктов `WHERE ... IN ()` или `WHERE ... NOT IN ()`, используйте методы **where_in** и **where_not_in** соответственно.

Оба метода принимают два аргумента. Первый - название столбца, с которым сравнивать. Второй - *массив* возможных значений. Как и во всех методах **where_**, вы можете указать множество столбцов, используя ассоциативный *массив* в качестве параметра.

```php
<?php
$people = ORM::for_table('person')->where_in('name', array('Fred', 'Joe', 'John'))->find_many();
```

4.5.9 Работа с NULL значениями: where_null и where_not_null

Для добавления пункта `WHERE column IS NULL` или `WHERE column IS NOT NULL`, используйте методы **where_null** и **where_not_null** соответственно. Оба метода принимают один параметр: название столбца для сравнения.

4.5.10 Необработанный WHERE

Если вам необходимо создать более сложный запрос, то можно использовать метод **where_raw** для указания нужного SQL-фрагмента для пункта WHERE. Данный метод принимает два аргумента: строку, добавляемую к запросу, и (опционально) массив параметров, который будет связан со строкой. Если параметры были переданы, строка должна содержать знаки вопроса (**?**) как плейсхолдеры, для подстановки вместо них значений из массива, а массив должен содержать значения, которые будут подставлены в строку в соответствующем порядке.

Данный метод можно использовать в цепочке методов вместе с другими методами **where_*** а так же с методами вроде **offset**, **limit** и **order_by_***. Содержимое переданной строки будет соединено с предыдущими и последующими пунктами WHERE с AND в качестве соединения.

```php
<?php
$people = ORM::for_table('person')
            ->where('name', 'Fred')
            ->where_raw('(`age` = ? OR `age` = ?)', array(20, 25))
            ->order_by_asc('name')
            ->find_many();

// Создаст SQL запрос:
SELECT * FROM `person` WHERE `name` = "Fred" AND (`age` = 20 OR `age` = 25) ORDER BY `name` ASC;
```

Обратите внимание, что этот метод поддерживает только синтакс “плейсхолдера в виде вопроса”, а НЕ синтакс “именной плейсхолдер”. Все потому, что PDO не позволяет создавать запросы, содержащие сме-

шанные типы плейсхолдеров. Так же, необходимо убедиться в том, что число вопросов-плейсхолдеров в строке соответствует числу элементов в массиве.

Если вам нужно ещё больше гибкости, вы можете вручную указать всю строку запроса. Смотрите *Необработанные запросы* ниже.

4.5.11 Limit и offset

*Обратите внимание, что эти методы **не* экранируют свои паораметры в запросе, поэтому они **не** должны передаваться напрямую от пользователя.*

Методы `limit` и `offset` очень похожи на эквивалентные им в SQL.

```
<?php
$people = ORM::for_table('person')->where('gender', 'female')->limit(5)->offset(10)->find_many();
```

4.5.12 Порядок

*Обратите внимание, что эти методы **не* экранируют свои паораметры в запросе, поэтому они **не** должны передаваться напрямую от пользователя.*

Доступны два метода для добавления к запросу пункта `ORDER BY`. Это `order_by_desc` и `order_by_asc`, каждый из которых принимает название столбца для сортировки. Имена столбцов будут писаться в кавычках.

```
<?php
$people = ORM::for_table('person')->order_by_asc('gender')->order_by_desc('name')->find_many();
```

Если вы хотите упорядочить по какому-то другому признаку, отличному от названия столбца, то используйте метод `order_by_expr` для добавления SQL выражения без кавычек, как в пункте `ORDER BY`.

```
<?php
$people = ORM::for_table('person')->order_by_expr('SOUNDEX(`name`)')->find_many();
```

4.6 Группировка

*Обратите внимание, что эти методы **не* экранируют свои паораметры в запросе, поэтому они **не** должны передаваться напрямую от пользователя.*

Для добавления пункта `GROUP BY` в строку запроса, вызовите метод `group_by` передав название столца в качестве аргумента. Можно вызывать этот метод множество раз для добавления большего числа колонок.

```
<?php
$people = ORM::for_table('person')->where('gender', 'female')->group_by('name')->find_many();
```

Так же возможно использование `GROUP BY` с выражениями из базы данных:

```
<?php
$people = ORM::for_table('person')->where('gender', 'female')->group_by_expr("FROM_UNIXTIME(`time`, '%Y-%m')")->fin
```

4.7 Having

При использовании агрегирующих функций в комбинации с `GROUP BY` вы можете использовать `HAVING` для фильтрации, относительно этих значений.

`HAVING` работает точно таким же способом, что и все методы `where*` в Idiorm. Замените `where_` на `having_` для использования этих функций.

Например:

```
<?php
$people = ORM::for_table('person')->group_by('name')->having_not_like('name', '%bob%')->find_many();
```

4.8 Столбцы в результате

По-умолчанию, все столбцы в выражении `SELECT` будут возвращены после запроса. То есть, вызывая:

```
<?php
$people = ORM::for_table('person')->find_many();
```

В результате сформирует запрос:

```
<?php
SELECT * FROM `person`;
```

Метод `select` дает контроль над тем, какие столбцы будут возвращены. Вызовите `select` несколько раз для указания нужных столбцов или используйте *select_ many* для указания нескольких столбцов за раз.

```
<?php
$people = ORM::for_table('person')->select('name')->select('age')->find_many();
```

В результате сформирует запрос:

```
<?php
SELECT `name`, `age` FROM `person`;
```

Опционально, можно передать второй аргумент в `select` для указания алиаса столбца:

```
<?php
$people = ORM::for_table('person')->select('name', 'person_name')->find_many();
```

В результате сформирует запрос:

```
<?php
SELECT `name` AS `person_name` FROM `person`;
```

Названия столбцов, переданные в `select` автоматически заносятся в кавычки, даже если содержат идентификаторы в стиле `table.column`:

```
<?php
$people = ORM::for_table('person')->select('person.name', 'person_name')->find_many();
```

В результате сформирует запрос:

```
<?php
SELECT `person`.`name` AS `person_name` FROM `person`;
```

Если вы хотите переопределить это поведение (например, для передачи выражения из базы данных) то вместо этого метода нужно использовать метод `select_expr`. Он так же принимает алиас в качестве второго аргумента. Можно указать несколько выражений, вызвав `select_expr` несколько раз или использовать *select_many_expr* для указания нескольких выражений за раз.

```
<?php
// Примечание: Только для иллюстрации. Для выполнения подсчется, используйте метод count().
$people_count = ORM::for_table('person')->select_expr('COUNT(*)', 'count')->find_many();
```

В результате сформирует запрос:

```
<?php
SELECT COUNT(*) AS `count` FROM `person`;
```

4.8.1 Короткая запись для указания множества столбцов

`select_many` и `select_many_expr` очень похожи, но позволяют указать более одного столбца за раз. Например:

```
<?php
$people = ORM::for_table('person')->select_many('name', 'age')->find_many();
```

В результате сформирует запрос:

```
<?php
SELECT `name`, `age` FROM `person`;
```

Для указания алиасов, вам нужно передать массив (алиас будет называться так же, как ключ в ассоциативном массиве):

```
<?php
$people = ORM::for_table('person')->select_many(array('first_name' => 'name'), 'age', 'height')->find_many();
```

В результате сформирует запрос:

```
<?php
SELECT `name` AS `first_name`, `age`, `height` FROM `person`;
```

Вы можете использовать такой стиль при передаче аргументов в `select_many` и `select_many_expr` путем смешивания и сопоставления массивов и параметров:

```
<?php
select_many(array('alias' => 'column', 'column2', 'alias2' => 'column3'), 'column4', 'column5')
select_many('column', 'column2', 'column3')
select_many(array('column', 'column2', 'column3'), 'column4', 'column5')
```

Все методы выборки могут быть соединены в цепочку друг с другом, так что вы могли сделать следующий аккуратный запрос на выборку включающий в себя выражения:

```
<?php
$people = ORM::for_table('person')->select_many('name', 'age', 'height')->select_expr('NOW()', 'timestamp')->find_m
```

В результате сформирует запрос:

```
<?php
SELECT `name`, `age`, `height`, NOW() AS `timestamp` FROM `person`;
```

4.9 DISTINCT

Для добавления ключевого слова `DISTINCT` перед списком результирующих столбцов в запросе, добавьте вызов метода `distinct()` в цепочку запроса.

```
<?php
$distinct_names = ORM::for_table('person')->distinct()->select('name')->find_many();
```

В результате сформирует запрос:

```
<?php
SELECT DISTINCT `name` FROM `person`;
```

4.10 Соединения Join

Idiorm содержит семейство методов для добавления различных типов `JOIN`(объединение) при построении запросов:

Методы: `join`, `inner_join`, `left_outer_join`, `right_outer_join`, `full_outer_join`.

Каждый из этих методов принимает одинковый набор аргументов. В описании будет использоваться базовый метод `join` в качестве примера, но то же самое применимо и к остальным методам.

Первые два аргумента являются обязательными. Первым идет имя таблицы для соединения, вторым условия для объединения. Рекумендуемым способом указания этих аргументов является *массив* содержащий три компонента: первый столбец, оператор, второй столбец. Название столбца и таблицы автоматический заключается в кавычки. Пример:

```
<?php
$results = ORM::for_table('person')->join('person_profile', array('person.id', '=', 'person_profile.person_id'))->f
```

Так же возможно указать условие в виде строки, которая будет добавлена в запрос как есть. Однако, в этом случае имена столбцов **не** будет экранированы, так что этот метод нужно использовать с осторожностью.

```
<?php
// Не рекомендуется, потому что условие объединения не экранируется.
$results = ORM::for_table('person')->join('person_profile', 'person.id = person_profile.person_id')->find_many();
```

Методы `join` так же принимают необязательный третий параметр, который служит как `alias` для таблицы в запросе. Это полезно, если нужно соединить таблицу с *собой* для создания иерархической структуры. В этом случае, лучше всего скомбинировать с методом `table_alias`, который добавит алиас к *основной* таблице, связанной с ORM, и методом `select` для управления возвращаемыми столбцами.

```
<?php
$results = ORM::for_table('person')
    ->table_alias('p1')
    ->select('p1.*')
    ->select('p2.name', 'parent_name')
    ->join('person', array('p1.parent', '=', 'p2.id'), 'p2')
    ->find_many();
```

4.10.1 Необработанные соединения JOIN

Если вам нужно построить более сложный запрос, то можно использовать метод `raw_join` для указания SQL-фрагмента для пункта JOIN. Этот метод принимает четыре обязательных аргумента: строку, добавляемую к запросу, условия в виде *массива*, содержащего три компонента: первый столбец, оператор, второй столбец, алиас таблицы и (необязательно) массив параметров. Если параметры переданы, строка должна содержать знаки вопроса (`?`) для представления подставляемых значений, и массив параметров должен создержать значения для подстановки в строку в корректной последовательности.

Этот метод может использоваться в цепочке наравне с другими методами `*_join` так же, как и методы вроде `offset`, `limit` и `order_by_*`. Содержимое переданной строки будет соединено с предшествующими и последующими пунктами JOIN.

```
<?php
$people = ORM::for_table('person')
            ->raw_join(
                'JOIN (SELECT * FROM role WHERE role.name = ?)',
                array('person.role_id', '=', 'role.id'),
                'role',
                array('role' => 'janitor'))
            ->order_by_asc('person.name')
            ->find_many();

// Создаст SQL запрос:
SELECT * FROM `person` JOIN (SELECT * FROM role WHERE role.name = 'janitor') `role` ON `person`.`role_id` = `role`.
```

Обратите внимание, этот метод поддерживает только синтаксис “плейсхолдеров в виде знака вопроса”, а не синтаксис “именованных плейсхоледров”. Потмоу что PDO не позволяет создавать запросы с различным типом плейсхоледров. Так же нужно убедиться, что число знаков вопроса в строке совпадает с числом передаваемых значений в массиве.

Если вам нужно ещё больше гибкости, вы можете вручную написать весь запрос. Смотрите *Необработанные запросы* ниже.

4.11 Агрегатные функции

Существует поддержка `MIN`, `AVG`, `MAX` и `SUM` в дополнение к `COUNT` (описанно ранее).

Для получения минимального значения столбца, вызовите метод `min()`.

```
<?php
$min = ORM::for_table('person')->min('height');
```

Остальные функции (`AVG`, `MAX` and `SUM`) работают таким же способом. Укажите название столбца для выполнения агрегатных функций, и в результате получите значение integer.

4.12 Необработанные запросы

Если вам нужно выполнять более сложные запросы, то можно полностью указать текст запроса, используя метод `raw_query`. Данный метод принимает строку и необязательный массив параметров. Строка может содержать плейсхолдеры, либо в виде знака вопроса, либо именованный плейсхолдер, который будет использоваться для внедрения параметров в запрос.

```php
<?php
$people = ORM::for_table('person')->raw_query('SELECT p.* FROM person p JOIN role r ON p.role_id = r.id WHERE r.nam
```

Возвращаемый экземпляр(ы) класса ORM будет содержать данные для всех столбцов, возвращаемых запросом. Обратите внимание, что все так же нужно вызывать метод `for_table` для связки экземпляра с нужной таблицей, даже если нет ограничений на установку другой таблицы в запросе. Это сделано для того, чтобы при желании выполнить метод `save`, ORM знал какую таблицу обновлять.

Нужно так же иметь в виду, что использование `raw_query` требует дополнительных навыков и может быть опасным, и Idiorm не будет пытаться защитить вас от ошибок, которые могут появиться при использовании этого метода. Если вы заметите у себя частое используете `raw_query`, то возможно вы неправильно поняли цель использования ORM, или ваше приложение может быть слишком сложным, для использоваия Idiorm. Рассмотрите возможность использования более полнофункциональной системы абстракции базы данных (ORM).

Глава 5

Модели

5.1 Получение данных из объектов

Итак, когда вы получили набор записей (объектов) в результате запроса, то можете получить доступ к свойствам у этих объектов (значениям, которые хранятся в столбцах соответствующих таблиц) двумя способами: используя метод `get`, или напрямую обращаясь к свойству объекта:

```php
<?php
$person = ORM::for_table('person')->find_one(5);

// Следующие два варианта записи являются эквивалетными
$name = $person->get('name');
$name = $person->name;
```

Так же есть возможность получить все данные внутри экземпляра ORM используя метод `as_array`. Он вернет ассоциативыний массив, описывающий имена столбцов (ключи) с их значениями.

Метод `as_array` принимает в качестве необязательного аргумента названия столбцов. Если был передан один или более таких аргументов, будут возвращены только совпадающие имена столбцов.

```php
<?php
$person = ORM::for_table('person')->create();

$person->first_name = 'Fred';
$person->surname = 'Bloggs';
$person->age = 50;

// Вернет array('first_name' => 'Fred', 'surname' => 'Bloggs', 'age' => 50)
$data = $person->as_array();

// Вернет array('first_name' => 'Fred', 'age' => 50)
$data = $person->as_array('first_name', 'age');
```

5.2 Обновление записей

Для обновления базы данных, измените одно или более свойств объекта, затем вызовите метод `save` для принятия измененных данных. Опять же, можно изменять значения свойств объектов используя метод `set` или напрямую указать значение свойства. Используя метод `set` можно так же обновить множество свойств за раз, передав в метод ассоциативный массив:

```php
<?php
$person = ORM::for_table('person')->find_one(5);

// Следующие два вида записи эквивалентны
$person->set('name', 'Bob Smith');
$person->age = 20;

// А это эквивалентно двум присваиваниям выше
$person->set(array(
    'name' => 'Bob Smith',
    'age'  => 20
));

// Синхронизируем объект с базой данных
$person->save();
```

5.2.1 Свойства, содержащие выражения

Можно задать свойства модели, содержащие выражения из базы данных, используя метод `set_expr`.

```php
<?php
$person = ORM::for_table('person')->find_one(5);
$person->set('name', 'Bob Smith');
$person->age = 20;
$person->set_expr('updated', 'NOW()');
$person->save();
```

Значение у `updated` столбца будет вставлено в запрос в чистом виде, позволяя тем самым базе данных выполнить любую нужную нам функцию - в данном случае `NOW()`.

5.3 Создание новых записей

Для добавления новой записи, сначала нужно создать "пустой" экземпляр объекта. Затем задать значения объекта как у обычного, и сохранить их.

```php
<?php
$person = ORM::for_table('person')->create();

$person->name = 'Joe Bloggs';
$person->age = 40;

$person->save();
```

После сохранения объекта, вы можете вызвать его метод `id()` для определения сгенерированного первичного ключа, который был присвоен базой данных.

5.4 Проверка, было ли изменено свойство

Для проверки, было ли свойство объекта изменено с тех пор, как он был создан (или последний раз сохранен), вызовите метод `is_dirty`:

```php
<?php
$name_has_changed = $person->is_dirty('name'); // Вернет true или false
```

5.5 Удаление записей

Для удаления объекта из базы данных, вызовите его метод `delete`.

```php
<?php
$person = ORM::for_table('person')->find_one(5);
$person->delete();
```

Для удаления более одного объекта из базы данных, постройте запрос:

```php
<?php
$person = ORM::for_table('person')
    ->where_equal('zipcode', 55555)
    ->delete_many();
```

Глава 6

Транзакции

Idiorm не поставляет каких-либо специальных методов для работы с транзакциями, но можно с легкостью использовать встронные в PDO меетоды:

```php
<?php
// Начало транзакции
ORM::get_db()->beginTransaction();

// Коммит транзакции
ORM::get_db()->commit();

// Откат транзакции
ORM::get_db()->rollBack();
```

Для более подробной информации, ознакомьтесь с документацией PDO по Transactions.

Глава 7

Множественные подключения

Idiorm теперь работает со множественными подключениями. Большинство статических функций работают с опциональным именем соединения в качестве дополнительного параметра. Касательно метода `ORM::configure`, это означает, что при передаче строки соединения для установки нового соединения, второй параметр, который обычно опускается, должен быть равен `null`. Во всех случаях, если имя соединения не предоставлено, то оно примет значение по-умолчанию `ORM::DEFAULT_CONNECTION`.

При составлении цепочки, как только метод `for_table()` был использован, оставшиеся вызовы в цепочке будут использовать корректное соединение.

```php
<?php
// Соединение по-умолчанию
ORM::configure('sqlite:./example.db');

// Именованное соединение, где 'remote' произвольное имя ключа
ORM::configure('mysql:host=localhost;dbname=my_database', null, 'remote');
ORM::configure('username', 'database_user', 'remote');
ORM::configure('password', 'top_secret', 'remote');

// Используя соединение по-умолчанию
$person = ORM::for_table('person')->find_one(5);

// Используя соединение по-умолчанию явным образом
$person = ORM::for_table('person', ORM::DEFAULT_CONNECTION)->find_one(5);

// Используя именованное соединение
$person = ORM::for_table('different_person', 'remote')->find_one(5);
```

7.1 Поддерживаемые методы

В каждом из этих случаев, параметр `$connection_name` необязательный, и является произвольным ключем, идентифицирующим название соединения.

- `ORM::configure($key, $value, $connection_name)`
- `ORM::for_table($table_name, $connection_name)`
- `ORM::set_db($pdo, $connection_name)`
- `ORM::get_db($connection_name)`
- `ORM::raw_execute($query, $parameters, $connection_name)`
- `ORM::get_last_query($connection_name)`

- `ORM::get_query_log($connection_name)`

Среди этих методов, только `ORM::get_last_query($connection_name)` *не* использует соединение по-умолчанию, если не передано название подключения. Вместо этого, если не передавать название подключения (или значение `null`) то метод вернет самый последний запрос *любого* соединения.

```
<?php
// Используя соединение по-умолчанию явным образом
$person = ORM::for_table('person')->find_one(5);

// Используя именованное подключение
$person = ORM::for_table('different_person', 'remote')->find_one(5);

// Последний запрос *любого* соединения
ORM::get_last_query(); // вернет запрос в 'different_person' используя имя 'remote'

// Вернет запрос 'person' используя переданную строку соединения из настроек по-умолчанию
ORM::get_last_query(ORM::DEFAULT_CONNECTION);
```

7.1.1 Примечания

- **Нет поддержки объединений (join) по подключениям**
- Множественные подключения не делятся настройками конфигурации. Это означает, что если у одного соединения параметр логов имеет значение `true` а у другого нет, то только запросы из соединения со включенным логом будут доступны для методов `ORM::get_last_query()` и `ORM::get_query_log()`.
- Был добавлен новый метод, `ORM::get_connection_names()`, возвращающий массив с именами соединений.
- Кэширование *должно* работать со множеством соединений (не забудьте включить кеширование для каждого соединения), однако это не надежно в отношении unit-тестирования. Пожалуйста, сообщайте об ошибках.

Глава 8

Индексы и таблицы

- genindex
- modindex
- search