Создаем новую таблицу в БД, используя миграции.

Синтаксис:

php artisan make:migration migration\_name --create="table\_name"

Нам надо создать таблицу litters с записями: id автоинкремент, integer, первичный ключ; litter – строковая, 2 символа .

php artisan make:migration create\_litters\_table --create="litters"

Дожидаемся подтверждения создания миграции.

Находим файл здесь\_будет\_дата\_и\_штамп\_времени \_create\_litters\_table.php миграции в папке database\migrations и начинаем его редактировать.

<?php

use Illuminate\Support\Facades\Schema;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Database\Migrations\Migration;

class CreateLittersTable extends Migration

{

public function up()

{

Schema::create('litters', function (Blueprint $table) {

$table->increments('id');

$table->string('litter', 2);

$table->timestamps();

});

}

public function down()

{

Schema::dropIfExists('litters');

}

}

И запускаем нашу миграцию командой

php artisan migrate

Если при применении миграции вы получаете ошибку «class not found» («класс не найден»), попробуйте выполнить команду composer dump-autoload и заново запустить команду migrate.

|  |
| --- |
| **Принудительные миграции в продакшене****:**  Некоторые операции миграций разрушительны, это значит что они могут привести к потере ваших данных. Для предотвращения случайного запуска этих команд на вашей боевой БД перед их выполнением запрашивается подтверждение. Для принудительного запуска команд без подтверждения используйте ключ --force:  php artisan migrate --force  Как обычно для unix систем просмотреть подсказки по доступным опциям команды migrate можно вызвав справку:  php artisan migrate –h  Вот еще полезные опции данной команды:  migrate:install Create the migration repository  migrate:refresh Reset and re-run all migrations  migrate:reset Rollback all database migrations  migrate:rollback Rollback the last database migration  migrate:status Show the status of each migration  [**Откат**](http://laravel.su/docs/5.0/migrations#rolling-back-migrations) **миграций:**  Отмена изменений последней миграции  php artisan migrate:rollback  Отмена изменений всех миграций  php artisan migrate:reset  Откат всех миграций и их повторное применение  php artisan migrate:refresh  php artisan migrate:refresh –seed |

Сразу создадим Модель для нашей таблицы:

php artisan make:model Litter

И отредактируем app\Litter.php

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Litter extends Model

{

protected $fillable = ["litter"];

}

Мы можем для тестирования использовать механизм наполнения таблиц БД начальными данными с помощью специальных классов. Они создаются и хранятся в директории database\seeds.

Создадим класс для добавления данных в только что созданную таблицу нашей БД.

php artisan make:seeder LitterTableSeeder

Теперь открываем новый класс в database\seeds\LitterTableSeeder.php

И правим класс, занося наши данные:

<?php

use Illuminate\Database\Seeder;

use App\Litter;

class LitterTableSeeder extends Seeder

{

public function run()

{

Litter::create(array('litter' => 'A'));

}

}

Теперь мы можем запустить наполнение нашей таблицы новыми данными, выполнив команду

php artisan db:seed --class=LittersTableSeeder

Но мы не будем так делать. Мы внесем изменения в уже имеющемся во фреймворке классе DatabaseSeeder, чтобы контролировать последовательное наполнение данными наши таблицы. Откроем для этого файл database\seeds\DatabaseSeeder.php и исправим его:

<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder

{

public function run()

{

$this->call('LitterTableSeeder');

$this->command->info(‘operation completed successfully!’);

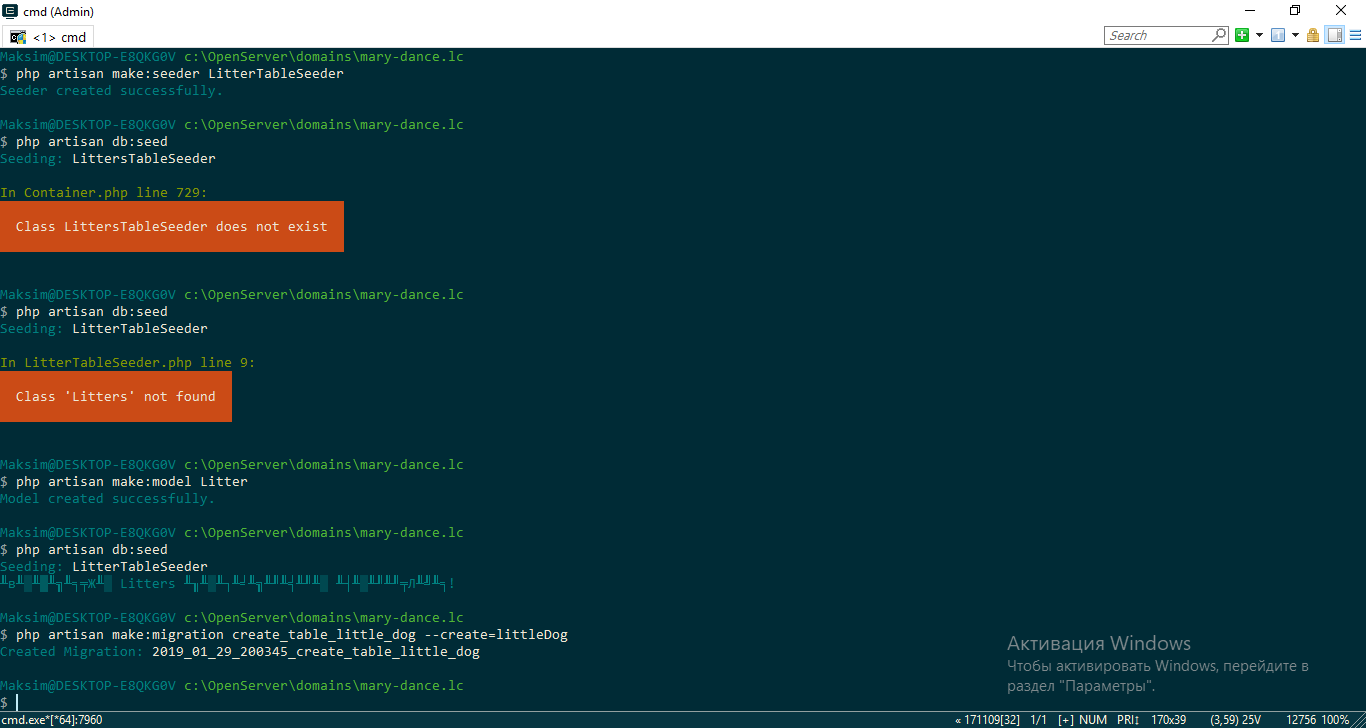
}

}

Сохраним его и запустим в командной строке:

php artisan db:seed

Это действие вызовет на исполнение класс DatabaseSeeder, в котором произойдет вызов класса LittersTableSeeder, произойдет наполнение данными, и выйдет сообщение о заполнении соответствующей таблицы. Если мы будем наполнять несколько таблиц, то необходимо создать отдельные классы для каждого действия по наполнению соответствующей таблицы, а потом указать вызовы этих классов в DatabaseSeeder в той последовательности, в которой мы хотим производить их наполнение.



<?php

use Illuminate\Support\Facades\Schema;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Database\Migrations\Migration;

class CreateTableLittleDog extends Migration

{

public function up()

{

Schema::create('littleDogs', function (Blueprint $table) {

$table->increments('id');

$table->integer('idLitter', false); //false указывает, что это не автоинкримент

$table->integer(‘idPhoto,

$table->timestamps();

});

}

public function down()

{

Schema::dropIfExists('littleDogs');

}

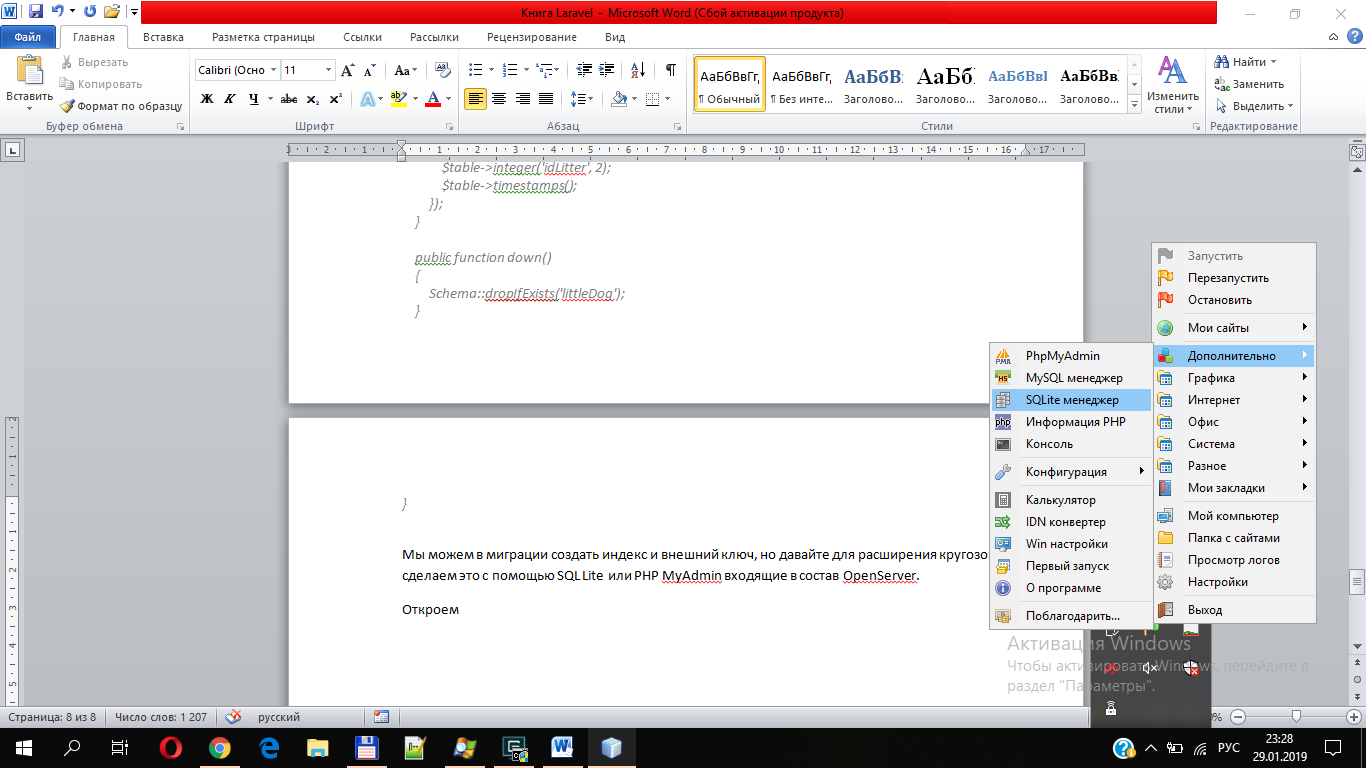
}

Создаем модель для таблицы

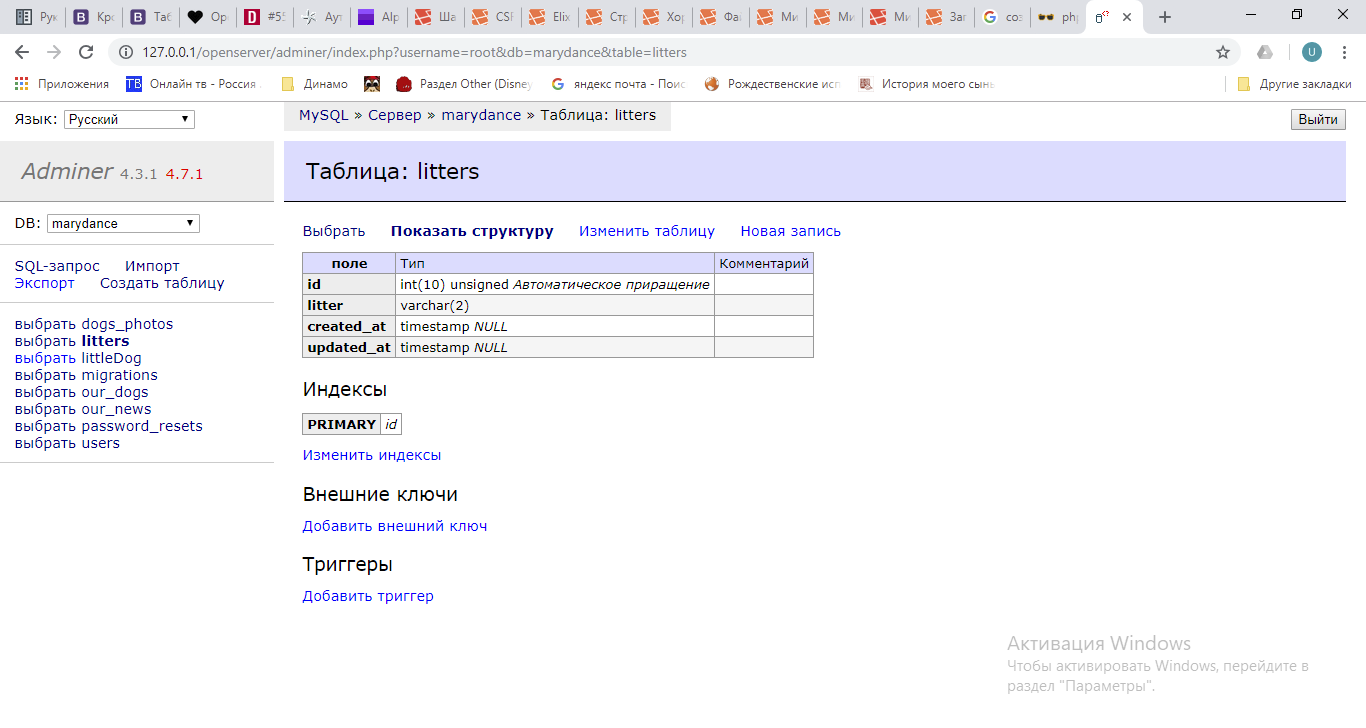
php artisan make:model littleDog

Мы можем в миграции создать индекс и внешний ключ, но давайте для расширения кругозора сделаем это с помощью SQLite или PHP MyAdmin входящие в состав OpenServer.

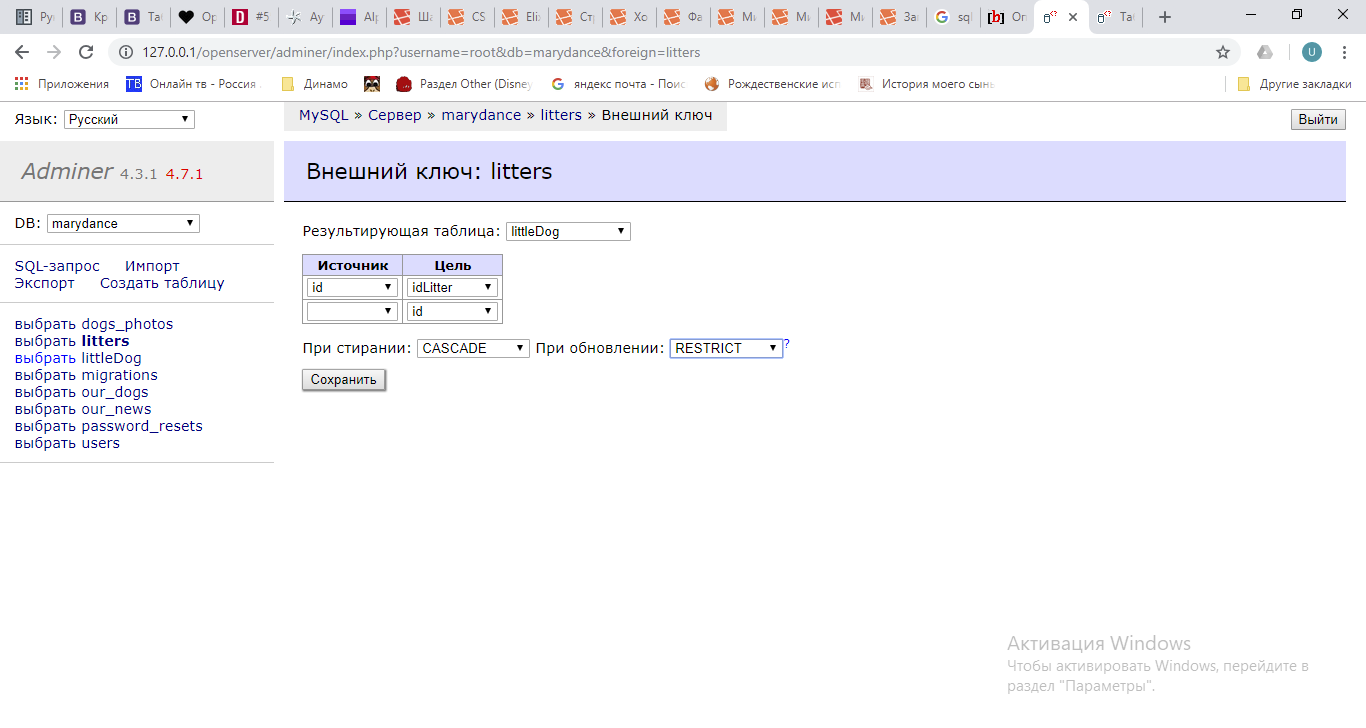
Откроем SQLite



Введя Логин/Пароль (врезка по первичной настройке OpenServer с БД – Логины и пароли по-умолчанию), выбираем нашу БД и нажимаем название таблицы



Нажимаем добавить внешний ключ и заполняем поля как на картинке



Для стирания (ON DELETE) выбрано CASCADE, т.е. удалять все связанные записи из дочерней таблицы, т.к. допустим если нам нужно удалить сам помет, то соответственно будут удалены и щенки, которые были в данном помете. Соответственно при обновлении (ON UPDATE) выбрано изменения и в соответствующих записях, чтобы не потерять целостность.