

Laravel

Урок 5. Работа с базой данных. Запросы. Миграции

Структура базы данных, соединение с БД, создание и наполнение таблиц. Миграции.

Оглавление

[Теория](#)

[Установка окружения.](#)

[.env-файл.](#)

[Конфигурация базы данных.](#)

[Практика](#)

[Создание миграций.](#)

[Seeding — посев данных.](#)

[Взаимодействие с базой данных.](#)

[Конструктор запросов.](#)

[Практическое задание](#)

[Дополнительные материалы](#)

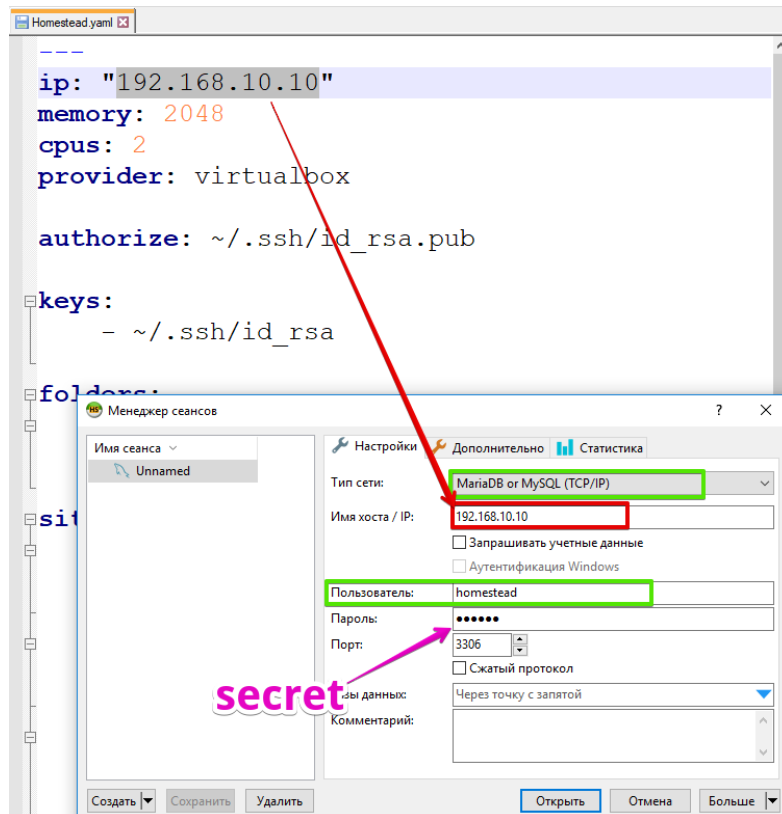
[Используемая литература](#)

Теория

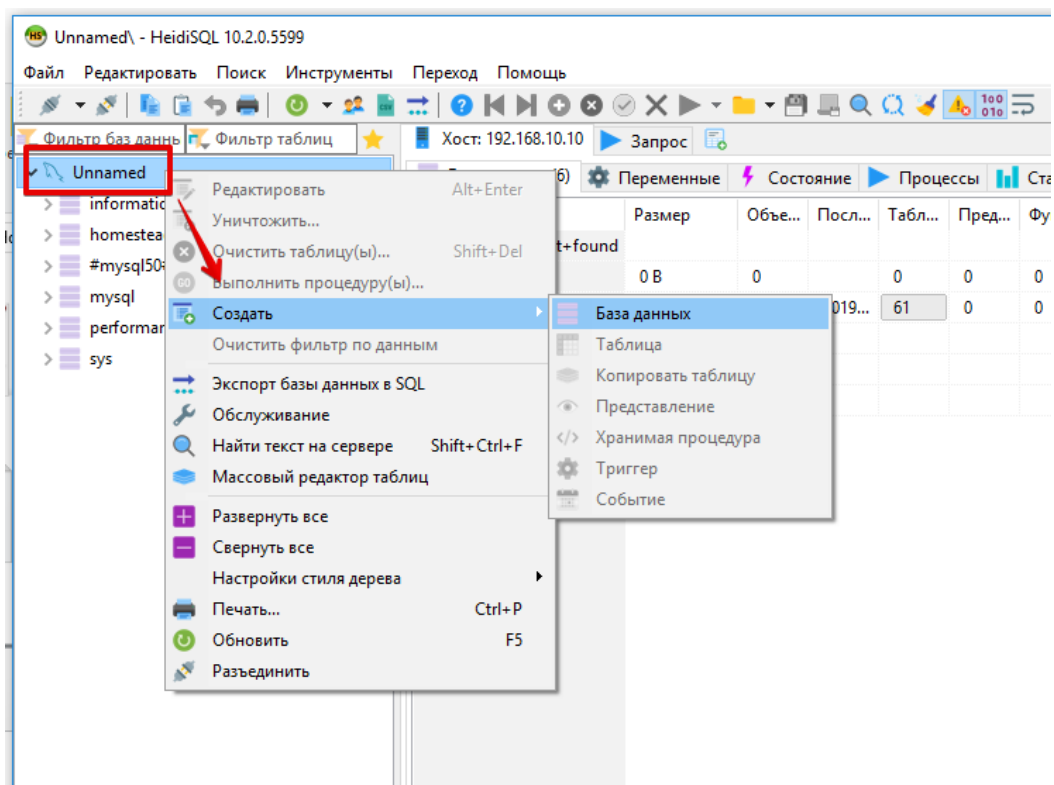
Установка окружения

Для работы с базой данных потребуется визуальный интерфейс HeidiSQL. Скачать эту программу можно с сайта <https://www.heidisql.com/download.php>

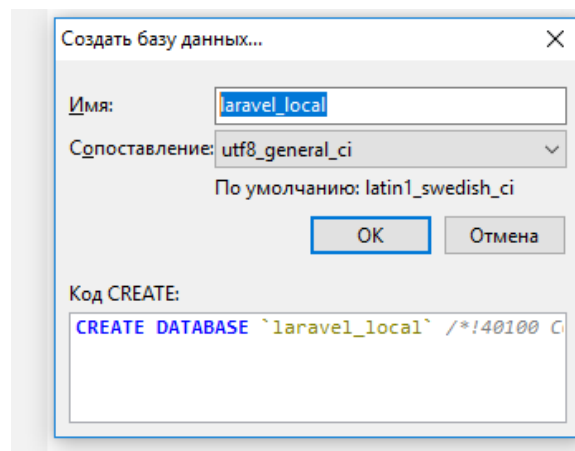
Установим и откроем программу. Выберем кнопку «Создать». Тип соединения — MySQL. IP-адрес укажем тот, что записан в настройках нашей виртуальной машины — **homestead**. Имя пользователя по умолчанию — **homestead**, введем его. Пароль — **secret**. Далее нажмем «Открыть».



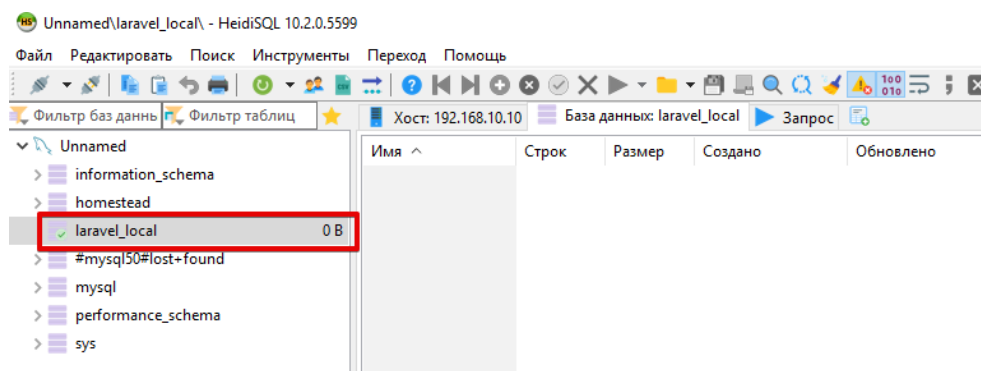
Теперь создаем базу данных. Для этого щелкаем правой кнопкой мыши по нашему соединению и выбираем «Создать» — «База данных».



В открывшемся окне вводим имя базы данных **laravel_local**. В поле «Сопоставление» выберем кодировку **utf8_general_ci**. Данные действия создадут команды SQL. Выполним их, нажав ОК.



Таким образом мы создали новую базу данных на нашем сервере.



.env-файл

Теперь, когда мы создали базу данных, самое время поговорить о переменных окружения Laravel. Они хранятся в файле `.env` в корневом каталоге Laravel. Но если мы зайдем в файл `.gitignore` в этом же каталоге, то обнаружим, что `.env` — игнорируемый файл и не будет записан в репозиторий.

Это нужно для безопасности. В репозитории не должны храниться эти данные. Однако мы должны указать, какие переменные будут использоваться в нашем приложении. Для этого в корневом каталоге Laravel используется файл `.env.example`.

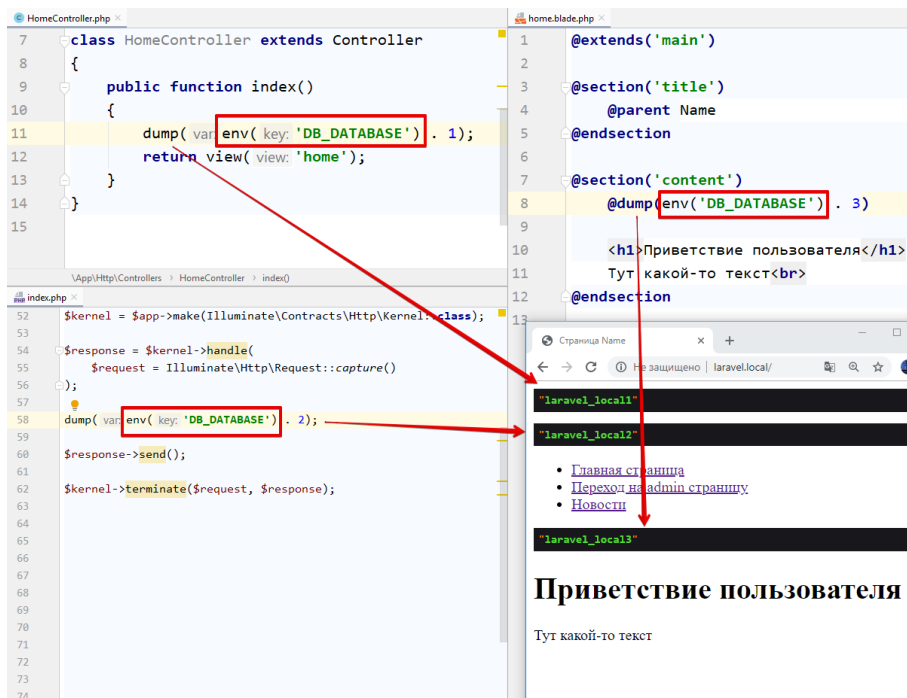
Добавим в `.env` информацию о созданной базе данных. Файл `.env.example` оставим без изменений.

```
Project
├── laravel
│   ├── app
│   ├── bootstrap
│   └── config
│       ├── app.php
│       ├── auth.php
│       ├── broadcasting.php
│       ├── cache.php
│       ├── database.php
│       ├── filesystems.php
│       ├── hashing.php
│       ├── logging.php
│       ├── mail.php
│       ├── queue.php
│       ├── services.php
│       ├── session.php
│       └── view.php
│   ├── database
│   ├── public
│   ├── resources
│   ├── routes
│   ├── storage
│   ├── tests
│   ├── vendor
│   ├── editorconfig
│   ├── .env
│   ├── .env.example
│   ├── .gitattributes
│   ├── .gitignore
│   ├── .styleci.yml
│   ├── artisan
│   ├── composer.json
│   ├── composer.lock
│   ├── package.json
│   ├── phpunit.xml
│   ├── readme.md
│   ├── server.php
│   └── webpack.mix.js
└── External Libraries
    └── Scratches and Consoles

.env
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:z+vMfuz98ufvxKiDq/qpIw8TVysxgRMelkUz/4BUDQY=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8
9 DB_CONNECTION=mysql
10 DB_HOST=127.0.0.1
11 DB_PORT=3306
12 DB_DATABASE=laravel_local
13 DB_USERNAME=homestead
14 DB_PASSWORD=secret
15
16 BROADCAST_DRIVER=log
17 CACHE_DRIVER=file
18 QUEUE_CONNECTION=sync
19 SESSION_DRIVER=file
20 SESSION_LIFETIME=120
21
22 REDIS_HOST=127.0.0.1
23 REDIS_PASSWORD=null
24 REDIS_PORT=6379
25
26 MAIL_DRIVER=smt
27 MAIL_HOST=smt.mailtrap.io
28 MAIL_PORT=2525
29 MAIL_USERNAME=null
30 MAIL_PASSWORD=null
31 MAIL_ENCRYPTION=null
32
33 AWS_ACCESS_KEY_ID=
34 AWS_SECRET_ACCESS_KEY=
35 AWS_DEFAULT_REGION=us-east-1
36 AWS_BUCKET=
37
38 PUSHER_APP_ID=
39 PUSHER_APP_KEY=
40 PUSHER_APP_SECRET=
41 PUSHER_APP_CLUSTER=mt1
42
43 MIX_PUSHER_APP_KEY="{PUSHER_APP_KEY}"
44 MIX_PUSHER_APP_CLUSTER="{PUSHER_APP_CLUSTER}"
45

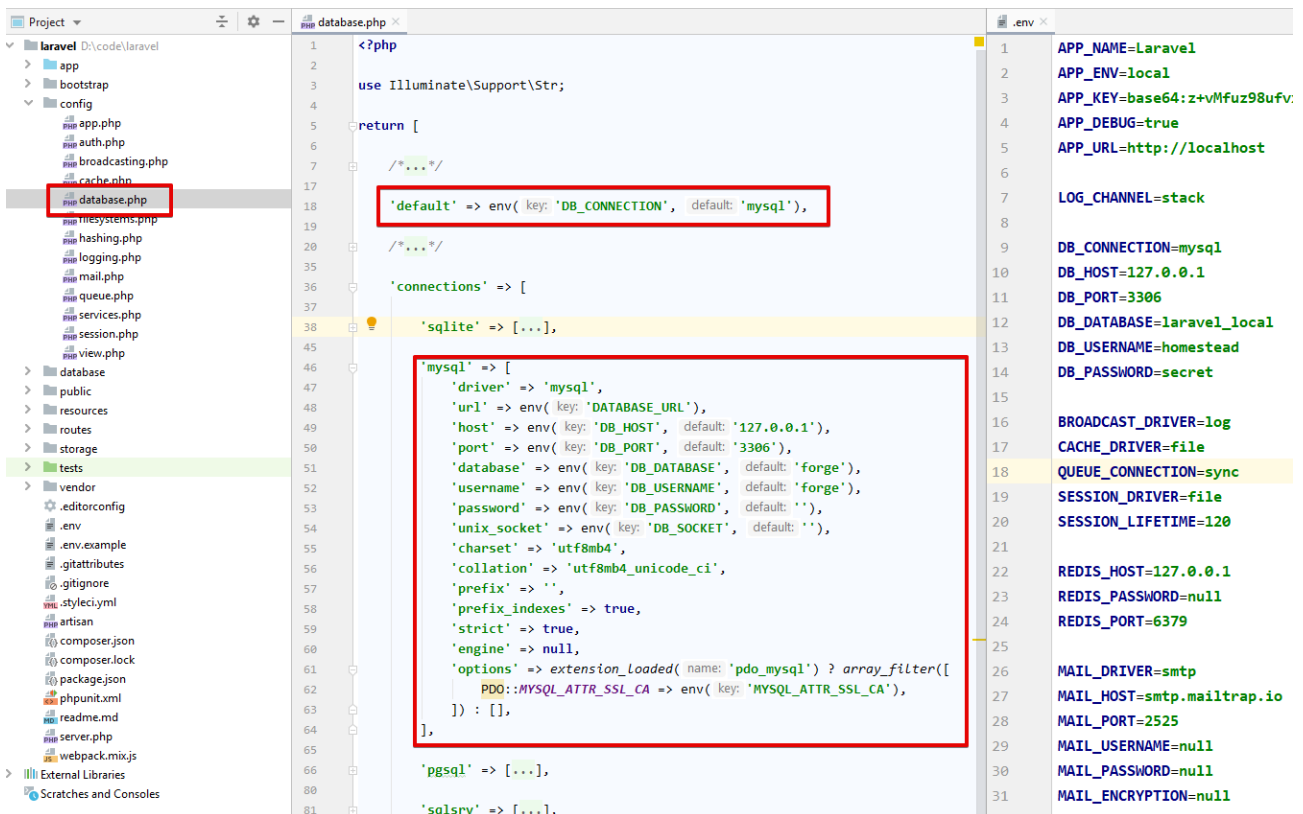
.env.example
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8
9 DB_CONNECTION=mysql
10 DB_HOST=127.0.0.1
11 DB_PORT=3306
12 DB_DATABASE=homestead
13 DB_USERNAME=homestead
14 DB_PASSWORD=secret
15
16 BROADCAST_DRIVER=log
17 CACHE_DRIVER=file
18 QUEUE_CONNECTION=sync
19 SESSION_DRIVER=file
20 SESSION_LIFETIME=120
21
22 REDIS_HOST=127.0.0.1
23 REDIS_PASSWORD=null
24 REDIS_PORT=6379
25
26 MAIL_DRIVER=smt
27 MAIL_HOST=smt.mailtrap.io
28 MAIL_PORT=2525
29 MAIL_USERNAME=null
30 MAIL_PASSWORD=null
31 MAIL_ENCRYPTION=null
32
33 AWS_ACCESS_KEY_ID=
34 AWS_SECRET_ACCESS_KEY=
35 AWS_DEFAULT_REGION=us-east-1
36 AWS_BUCKET=
37
38 PUSHER_APP_ID=
39 PUSHER_APP_KEY=
40 PUSHER_APP_SECRET=
41 PUSHER_APP_CLUSTER=mt1
42
43 MIX_PUSHER_APP_KEY="{PUSHER_APP_KEY}"
44 MIX_PUSHER_APP_CLUSTER="{PUSHER_APP_CLUSTER}"
45
```

Как получить доступ к переменным окружения? Для этого в Laravel используется специальный хелпер `env`, который доступен в любой части приложения. Он принимает несколько параметров. Первый (обязательный) — имя переменной окружения. Второй — значение по умолчанию, если переменная не определена.



Конфигурация базы данных

Для конфигурации базы данных используется файл **database.php**, находящийся в каталоге **config**. Открыв этот файл, увидим значение по умолчанию используемой базы данных — **mysql**. Ниже есть настройки, содержащиеся в массиве **connections**. Данный массив собирает значение из файла **.env** при помощи хелпера **env**.



Практика

Создание миграций

У приложения есть жизненный цикл, во время которого производятся дополнения и правки функциональности. Поэтому в приложении часто требуется изменение базы данных. Прямое изменение — не лучший вариант, поскольку оно не исключает человеческий фактор: есть риск ошибки. Поэтому чаще всего используются миграции, чтобы гарантировать одинаковые изменения на тестовом сервере и на боевом, который используется в продакшене. Еще один плюс миграции: как правило, их можно отменить к предыдущей версии базы. Это удобно при обнаружении ошибок, не выявленных на этапе разработки и тестирования.

Создавать миграции мы будем при помощи консоли. Сначала проверим список опций команды по созданию миграции. Для этого в консоли вводим **php artisan help make:migration**:

```
vagrant@homestead: ~/code/laravel
vagrant@homestead:~/code/laravel$ php artisan help make:migration
Description:
  Create a new migration file

Usage:
  make:migration [options] [--] <name>

Arguments:
  name                The name of the migration

Options:
  --create[=CREATE]  The table to be created
  --table[=TABLE]    The table to migrate
  --path[=PATH]      The location where the migration file should be created
  --realpath          Indicate any provided migration file paths are pre-resolved absolute paths
  --fullpath         Output the full path of the migration
  -h, --help         Display this help message
  -q, --quiet         Do not output any message
  -V, --version      Display this application version
  --ansi             Force ANSI output
  --no-ansi          Disable ANSI output
  -n, --no-interaction Do not ask any interactive question
  --env[=ENV]        The environment the command should run under
  -v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug
vagrant@homestead:~/code/laravel$
```

Создадим новую таблицу для сохранения новостей. Для этого введем команду: **php artisan make:migration create_news_table --create=news**:

```
vagrant@homestead: ~/code/laravel
vagrant@homestead:~/code/laravel$ php artisan make:migration create_news_table --create=news
Created Migration: 2019_09_06_083501_create_news_table
vagrant@homestead:~/code/laravel$
```

Создался новый файл в директории **~/database/migrations**. Открыв этот файл, увидим два метода: **up**, который будет вызываться во время наката (выполнения) миграций, и **down**, который будет выполняться, если придется откатить миграцию.

```
5 use ...
6
7 class CreateNewsTable extends Migration
8 {
9     /**
10      * Run the migrations.
11      *
12      * @return void
13      */
14     public function up()
15     {
16         Schema::create( table: 'news', function (Blueprint $table) {
17             $table->bigIncrements( column: 'id');
18             $table->timestamps();
19         });
20     }
21
22     /**
23      * Reverse the migrations.
24      *
25      * @return void
26      */
27     public function down()
28     {
29         Schema::dropIfExists( table: 'news');
30     }
31 }
32
```

Стоит отметить, что создание файла миграций не означает его выполнение. Как выполнять миграции, посмотрим позже.

Теперь подробнее рассмотрим, что содержится в методе **up**. Видим фасад **Schema** — обертку для класса `\Illuminate\Database\Schema\Builder`. В этом методе происходит обращение к **create** — статическому методу фасада. А фактически выполняется метод у объекта класса `\Illuminate\Database\Schema\Builder`. Посмотрим внимательнее на этот и другие методы объекта данного класса.

```

64  /** Determine if the given table exists. ...*/
70  public function hasTable($table){...}
78
79  /** Determine if the given table has a given column. ...*/
86  public function hasColumn($table, $column){...}
92
93  /** Determine if the given table has given columns. ...*/
100 public function hasColumns($table, array $columns){...}
112
113 /** Get the data type for the given column name. ...*/
120 public function getColumnType($table, $column){...}
126
127 /** Get the column listing for a given table. ...*/
133 public function getColumnListing($table){...}
141
142 /** Modify a table on the schema. ...*/
149 public function table($table, Closure $callback){...}
153
154 /** Create a new table on the schema. ...*/
161 public function create($table, Closure $callback){...}
169
170 /** Drop a table from the schema. ...*/
176 public function drop($table){...}
182
183 /** Drop a table from the schema if it exists. ...*/
189 public function dropIfExists($table){...}
195
196 /** Drop all tables from the database. ...*/
203 public function dropAllTables(){...}
207
208 /** Drop all views from the database. ...*/
215 public function dropAllViews(){...}
219
220 /** Drop all types from the database. ...*/
227 public function dropAllTypes(){...}
231
232 /** Rename a table on the schema. ...*/
239 public function rename($from, $to){...}

```

Первым параметром метода **create** передается строка — название будущей таблицы. Вторым — анонимная функция. Она в качестве параметра принимает объект класса **Illuminate\Database\Schema\Blueprint**, который будет подставлен в переменную **\$table** при помощи **Dependency Injection(DI)**. Благодаря методам этого объекта мы и создаем нужную структуру таблицы внутри самой функции.

У этого класса много методов, подробнее о них: <https://laravel.com/docs/5.8/migrations#columns>.

Разберем некоторые из них:

- **bigIncrements** — создает столбец первичного ключа с автоинкрементом. Принимает в качестве параметра строку, которая и служит названием столбца;
- **timestamps** — добавляет в таблицу два столбца: **created_at** и **updated_at**;
- **string** — добавляет в таблицу столбец с типом данных **VARCHAR**. Первым параметром принимает строку, которая будет именем столбца. Второй параметр необязательный — длина строки;
- **text** — добавляет в таблицу столбец с типом данных **TEXT**. Первым параметром принимает строку, которая будет именем столбца;
- **json** — добавляет в таблицу столбец с типом данных **JSON**. Первым параметром принимает строку, которая будет именем столбца;

- **bigInteger** — добавляет в таблицу столбец с типом данных **BIGINT**. Первым параметром принимает строку, которая будет именем столбца;
- **dateTime** — добавляет в таблицу столбец с типом данных **DATETIME**. Первым параметром принимает строку, которая будет именем столбца;
- **boolean** — добавляет в таблицу столбец с типом данных **BOOLEAN**. Первым параметром принимает строку, которая будет именем столбца.

Помимо основных методов, которые определяют тип данных столбца в таблице, есть модификаторы колонок, которые позволяют установить дополнительные параметры столбца:

- **default** — метод указывает на то значение, которое будет установлено для колонки по умолчанию;
- **nullable** — метод принимает **true** || **false** в качестве параметра и указывает, может ли быть значение в данной колонке **null**.
- **comment** — принимает в качестве параметра строку, которая станет комментарием для указанного столбца.

Данные модификаторы вызываются после определения типа столбца.

Добавим в метод **up** — код, который создаст нашу таблицу:

```

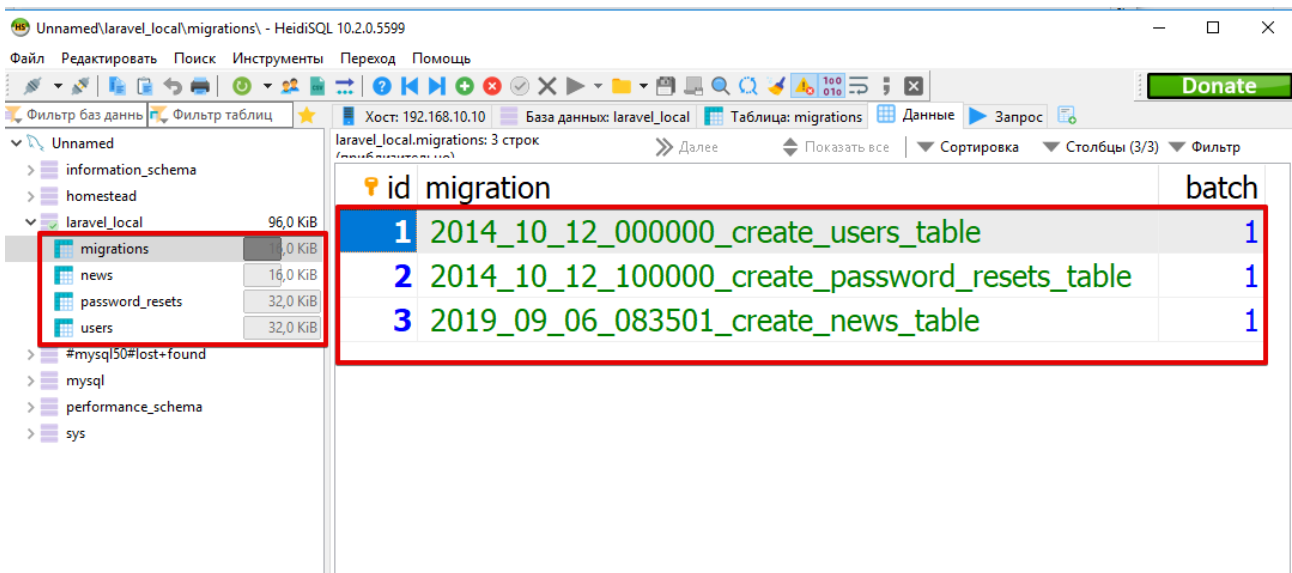
2
3 use Illuminate\Support\Facades\Schema;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Database\Migrations\Migration;
6
7
8 class CreateNewsTable extends Migration
9 {
10     /**
11      * Run the migrations.
12      *
13      * @return void
14      */
15     public function up()
16     {
17         Schema::create( table: 'news', function (Blueprint $table) {
18             $table->bigIncrements( column: 'id');
19             $table->string( column: 'title')->comment( comment: 'Заголовок');
20             $table->text( column: 'inform')->comment( comment: 'Содержание новости');
21             $table->boolean( column: 'is_private')
22                 ->default( value: true)
23                 ->comment( comment: 'Доступно ли содержание новости для неавторизованных');
24         });
25     }
26
27     /**
28      * Reverse the migrations.
29      *
30      * @return void
31      */
32     public function down()
33     {
34         Schema::dropIfExists( table: 'news');
35     }
36 }

```

Для выполнения миграций выполним команду **php artisan migrate**:

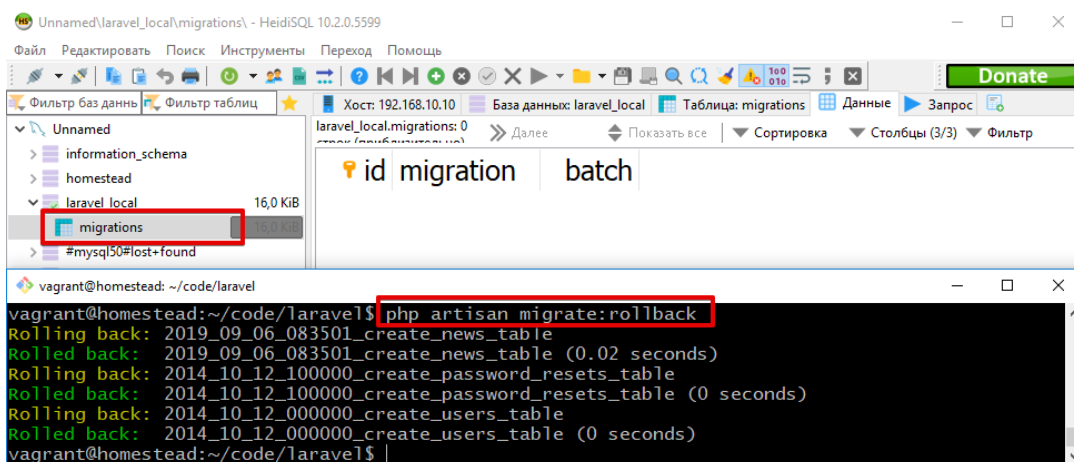
```
vagrant@homestead: ~/code/laravel
vagrant@homestead:~/code/laravel$ php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (0.08 seconds)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (0.11 seconds)
Migrating: 2019_09_06_083501_create_news_table
Migrated: 2019_09_06_083501_create_news_table (0.06 seconds)
vagrant@homestead:~/code/laravel$
```

Обратим внимание: мы выполнили все миграций, которые были в папке **migrations**. Перейдем в программу HeidiSQL и посмотрим, какие таблицы были созданы.

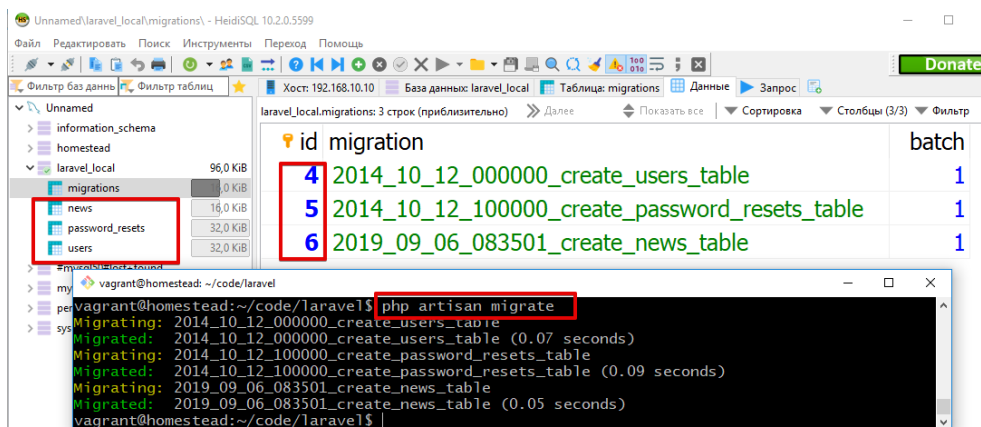


В нашей базе данных появилось четыре таблицы. Рассмотрим подробнее таблицу **migrations**. В этой таблице хранится информация о тех миграциях, которые были выполнены. Глядя на данные в ней, можем заметить, что сначала была создана таблицы **users**, затем таблица **password_resets** и после этого наша таблица для хранения новостей.

Если запустим миграцию еще раз предыдущей командой, то миграции, которые уже были выполнены, выполняться не будут. Но мы можем откатить эти миграции, выполнив команду **php artisan migrate:rollback**:



Данная команда откатит все миграции, которые были выполнены в данном предложении. На скриншоте видно, что все таблицы были удалены, но таблица **migrations** осталась. Повторим миграции.



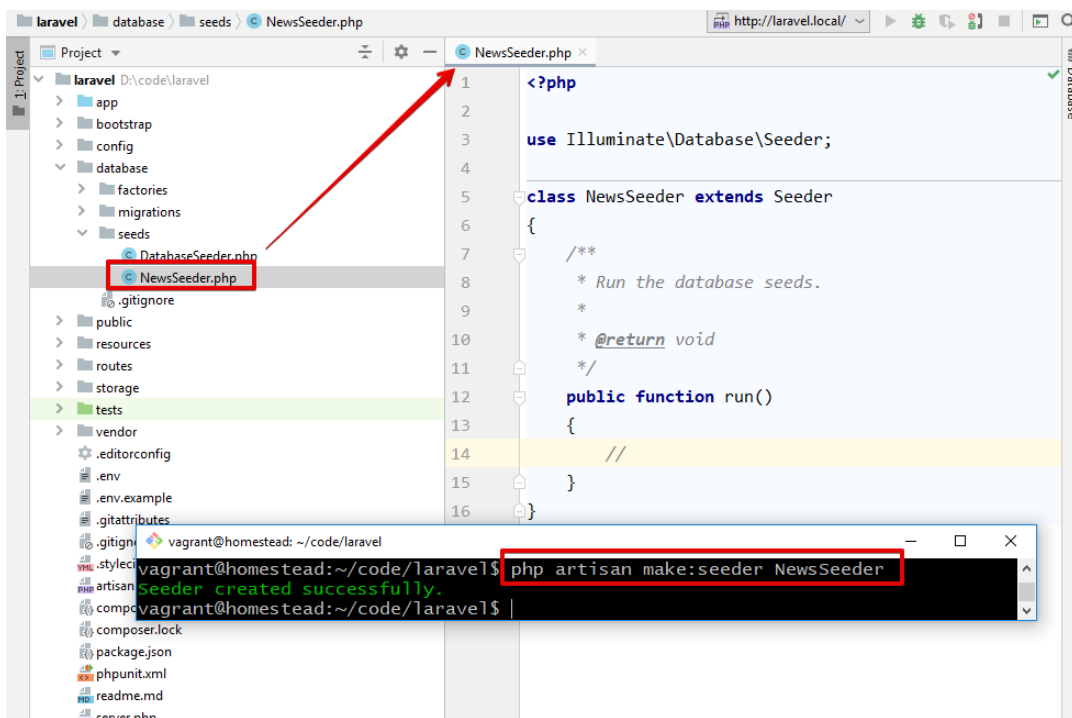
В том случае, если нам нужно откатить только последнюю миграцию или две последних миграции, то можем воспользоваться командой: `php artisan migrate:rollback --step=2`. Количество шагов в данной команде указывает какое количество миграций следует откатить.

Seeding — посев данных

Иногда для тестирования приложения необходимо какие-то входные данные. Их можно занести вручную, но это не совсем удобно. Удобнее если эти данные добавляются автоматически. Laravel предлагает для этой задачи решение, которое называется seeder. Их принцип действия похож на работу миграций — разработчик добавляет в проект специальный класс, в котором хранится логика для добавления данных.

У нас сейчас создана таблица News. Запомним ее данными, и для этого создадим новый seeder, выполнив команду: `php artisan make:seeder NewsSeeder`.

После выполнения данной команды в папке `~database/seeds` будет создан новый файл `NewsSeeder.php` — файл одноименного класса, в котором содержится единственный метод `run`, который будет выполняться при запуске seeder.



Для заполнения базы данных фейковой информацией воспользуемся библиотекой **fzaninotto/Faker**. Предварительно установим ее через **composer** — команда для установки библиотеки: **composer require fzaninotto/faker**.

Подробная информация по данной библиотеке: <https://github.com/fzaninotto/Faker>

Добавим в **seeder** немного кода. На скриншоте ниже в методе **run** обращаемся к фасаду **DB**, который обратится к объекту класса **Illuminate\Database\Query\Builder**. Таким образом мы получим возможность добавлять данные в указанную таблицу **news**. У полученного объекта вызываем метод **insert**, который и будет производить данную вставку. Метод **insert** ожидает массив данных, в котором ключи будут соответствовать полям таблицы.

Для генерации данных создаем отдельный метод **getData** и в нем обращаемся к библиотеке, которую установили. Обращаемся к статическому методу **create** и указываем локализацию. После этого набираем массив из 10 подмассивов. Каждый подмассив — отдельная строка в базе данных.

Для каждого подмассива собираем фейковые данные. Для этого используем объект, сохраненный в переменной **\$faker**, и вызываем соответствующие метод:

- **sentence** — метод класса **Faker\Generator**, возвращающий предложение, состоящее из указанного количества строк. В данной библиотеке нет реализации данного метода на русском языке, поэтому результат на английском;
- **realText** — возвращает указанное количество слов из реального текста, минимум 10 и максимум 200.

```
<?php
1
2
3 use Illuminate\Database\Seeder;
4 use Illuminate\Support\Facades\DB;
5
6 class NewsSeeder extends Seeder
7 {
8     /** Run the database seeds. ...*/
9
10    public function run()
11    {
12        DB::table( table: 'news' )->insert( $this->getData() );
13    }
14
15    private function getData()
16    {
17        $faker = Faker\Factory::create( locale: 'ru_RU' );
18
19        $data = [];
20        for ( $i = 0; $i < 10; $i++ ) {
21            $data[] = [
22                'title' => $faker->sentence( rand( 3, 10 ) ),
23                'inform' => $faker->realText( rand( 100, 200 ) ),
24                'is_private' => (boolean)rand( 0, 1 ),
25            ];
26        }
27        return $data;
28    }
29 }
30
31
32
33
34
35
36
37
```

Для запуска посева необходимо в консоли выполнить команду `php artisan db:seed --class=NewsSeeder`.

Seeders можно запускать любое количество раз.

The screenshot shows a database interface with a table named 'news' in the 'laravel_local' database. The table has 30 rows of data. A terminal window in the foreground shows the command 'php artisan db:seed --class=NewsSeeder' being executed three times, with the output 'Database seeding completed successfully.' visible each time.

| id | title | inform |
|----|--|--|
| 1 | Facere atque quaerat provident modi culp... | Вообрази, что в доме есть много других занятий, кроме продолжительных по... |
| 2 | Laborum aspernatur et non voluptas non ... | Да кто вы такой? — сказал белокуроый. — Не хочю. — Ну да ведь меня — всю св... |
| 3 | Laudantium aperiam qui adipisci. | Потом пили какой-то бальзам, носивший такое имя, которое даже трудно был... |
| 4 | Accusamus voluptas aut est quidem. | Это все выдмали доктора немцы да французы, я бы их — перевешал за это! Т... |
| 5 | Fuga molestiae accusantium nihil ex repell... | Чичикову; Чичиков заметил, однако же, как-то вскользь, что самому себе он н... |
| 6 | Itaque dolores id rerum. | Но как ни в чем не думал, как только о постели. Не успела бричка совершенно... |
| 7 | Delentit quod voluptatem velit incidunt bla... | Маниловка, может быть, так же было — пятьдесят. Ферарди четыре часа верт... |
| 8 | Est modi quis culpa accusantium optio velit. | Последнего заключения Чичиков никак не подумал, — продолжал Собакевич, |
| 9 | Nam iure quibusdam repudiandae. | Манилов. — впрочем, приезжаем в город — для препровождения времени, де... |
| 10 | Rerum aliquam nemo in asperiores. | Ноздрев, скорее за шапку да по-за спиною капитана-исправника выскользнул н... |
| 11 | Est nemo eaque et eos. | Написавши письмо, дал он ей подписаться и попросил маленький списокчек му... |
| 12 | Autem saepe nihil magni sint. | Ноздрева. Увы! несправедливы будут те, которые станут говорить так. Ноздрев... |
| 13 | Sed quia eos. | Куда ж? — сказал Чичиков. — О! Павел Иванович, — сказал наконец Собакевич... |
| 21 | Vel expedita enim provident ipsum aut n... | ведь я — отгадал бы все, то есть те души, которые, точно, уже умерли. Мани... |
| 22 | Dolor ducimus fugit eum accusamus corpo... | Она была недурна, одета к лицу. На ней хорошо сидел матерчатый шелковый к... |
| 23 | Quia enim voluptatem. | А! теперь хорошо! прощайте, матушка! Коня тронулись. Селифан был во всю п... |
| 24 | Ex ex a qui libero enim. | Сейчас видно, — что двуличный человек! — Губернатор превосходный челове... |
| 25 | Accusamus dolorem quidem maiores occa... | Покой был известного рода, то есть без земли? — Нет, барин, как можно, что... |
| 26 | Quo consequatur voluptatem delectus co... | Ван понадобились души, я и так же было — хорошео, если бы, например, тако... |
| 27 | Vel maxime sed perferendis voluptate. | Или вы думаете, сыщите такого дурака, который бы хотя одним чином был его... |
| 28 | Cum nostrum rerum fugiat aut quia minus ... | Ну, душа, вот это так! Вот это тебе и есть порядочный человек: — прокурор; ; |
| 29 | Veritatis doloremque quisquam voluptatib... | Между тем три экипажа поджатили уже к крыльцу дома Ноздрева. В доме не б... |
| 30 | voluptatibus iste itaque aut voluptatum. | Дождь, однако же, давно нет на свете; но Собакевич так сказал утвердительно... |

Взаимодействие с базой данных

Теперь, когда у нас есть база данных, а в ней — необходимые нам данные, реализуем получение этих данных, их вставку и удаление.

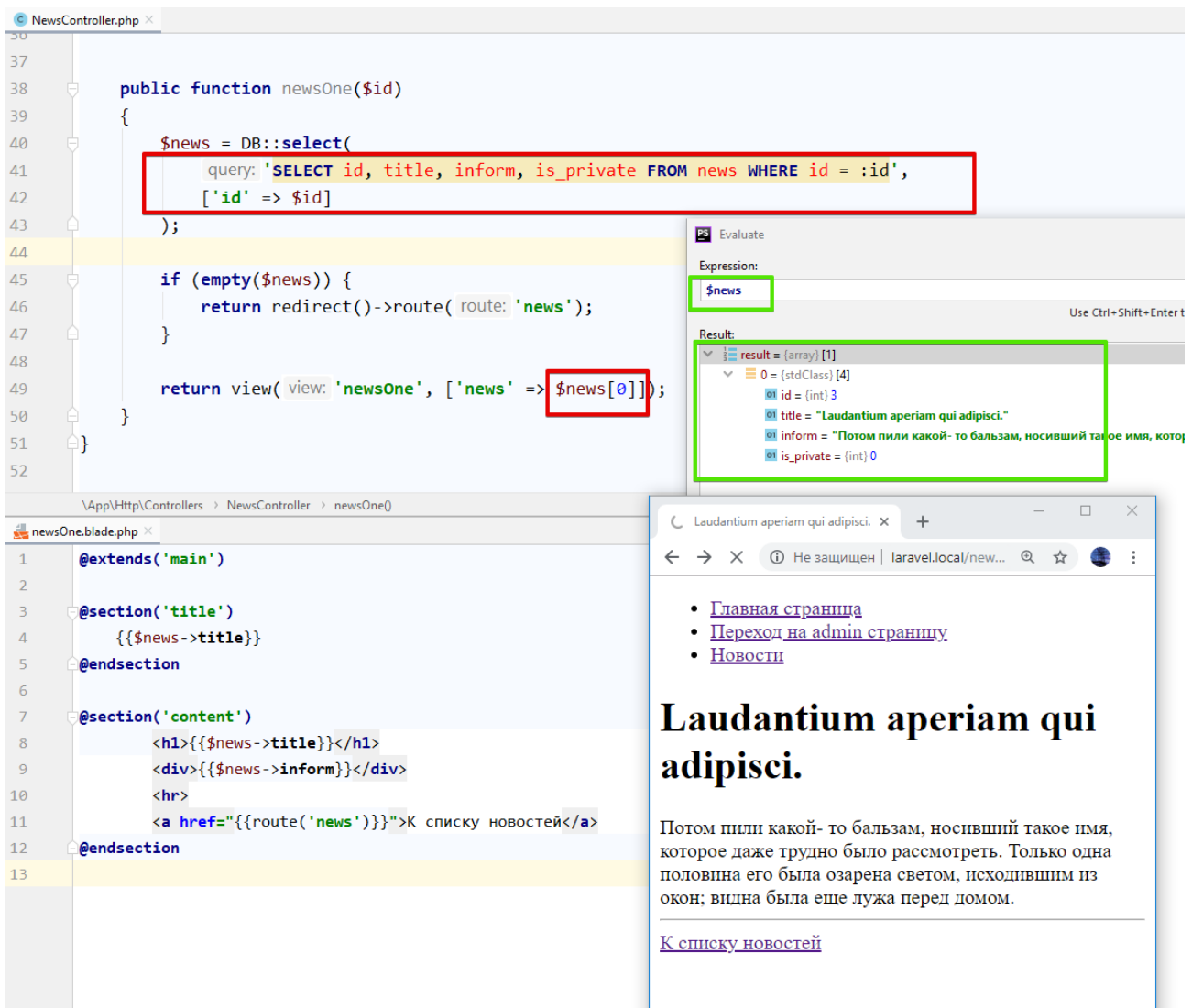
Для взаимодействия с базой данных воспользуемся фасадом **DB** и его методами. Первый метод, который мы применим, **select**. Он позволяет использовать обычные SQL-запросы с возможностью экранировать данные, полученные от пользователя.

The screenshot displays the following components:

- NewsController.php:** Line 28 shows a database query: `$news = DB::select('SELECT id, title, inform, is_private FROM news');`
- news.blade.php:** A Blade template with sections for title and content. A `@forelse` loop iterates over `$news`. Inside the loop, `@dump($item)` is used to debug, and an `@if` statement checks for `is_private` status.
- Evaluate Window:** Shows the execution of the SQL query, returning an array of 30 items. The first item is a `stdClass` object with properties: `id` (1), `title` ("Facere atque quaerat provident modi culpa possimus impedit ut minus dolore."), `inform` ("Вообрази, что в доме есть много других занятий, кроме продолжит..."), and `is_private` (1).
- Browser Preview:** Shows the rendered HTML page with a navigation menu, a heading "Новости", and the first news item's content.

Как видно на скриншоте, в **NewsController** используется выборка из базы данных. Статический метод **Select** принимает SQL-запрос, в результате которого получаем массив **std**-классов. **stdClass** в PHP является predefined и часто используется для заполнения динамическими свойствами. В нашем случае каждый класс в полученном массиве содержит поля в соответствии с указанными в SQL-запросе. Учитывая это, следует произвести изменения в шаблоне **news.blade.php**. Данные изменения также показаны на скриншоте выше.

Теперь в методе **newsOne** в **NewsController** внесем изменения, чтобы получить конкретную запись из базы данных, а также создадим новый шаблон для вывода этой информации.



На скриншоте видим, что в метод **Select** передаем теперь два параметра. Первым передается SQL. Обратим внимание на то, что в данном запросе используется `:id`. Логично было бы подставить в эту строку переменную `$id`, которая и содержит идентификатор нужной записи. Но надо помнить, что эта переменная заполняется из данных, переданных пользователем. А значит, данные не безопасны — они могут содержать SQL-инъекцию. Подробнее о ней можно прочитать в статье <https://habr.com/ru/post/148151/>. Если в двух словах, это атака на базу данных.

Для защиты SQL-инъекций в Laravel используются подготовленные запросы. Сначала базе говорится, какой запрос будет происходить, и указывается, куда в запросе нужно подставить данные. Затем передаются данные и выполняется запрос. Данные подставляются в строго отведенные им места, что исключает атаку.

Вторым параметром в **Select** передаем массив, ключом в котором указываем названия плейсхолдеров. В значении передаем данные для подстановки. В примере выше в SQL-запросе используется один плейсхолдер `:id`, поэтому в массиве содержится одна пара «ключ — значение»: `'id' => $id`.

Помимо статичного метода для получения данных из базы, у фасада **DB** есть и другие:

- `selectOne(string $query, array $bindings = [])` — метод для поиска одной записи;

- `select(string $query, array $bindings = [])` — метод для выборки более чем одной записи;
- `insert(string $query, array $bindings = [])` — метод для вставки;
- `update(string $query, array $bindings = [])` — метод для изменения;
- `delete(string $query, array $bindings = [])` — метод для удаления;
- `statement(string $query, array $bindings = [])` — метод для запросов, не возвращающих результат.

Конструктор запросов

В практике работы с базой данных использование подхода, описанного выше, допустимо. Но он используется реже, чем конструктор запросов. Рассмотрим следующий пример:

```

31 public function news()
32 {
33     $news = DB::table('news')->get();
34     return view('news', ['news' => $news]);
35 }
36
37 public function newsOne($id)
38 {
39     $news = DB::table('news')->find($id);
40
41     if (empty($news)) {
42         return redirect()->route('news');
43     }
44
45     return view('newsOne', ['news' => $news]);
46 }
47 }

```

Здесь используется фасад DB, у которого вызывается метод `table`. Он принимает в качестве параметра название таблицы, к которой будет выполнен запрос. В данном случае обращаемся к таблице `news`. В результате метод `table` возвращает объект класса `Illuminate\Database\Query\Builder`. Он содержит большое количество методов, которые используются для создания запросов к базе данных.

```

2083
2084  /** Execute a query for a single record by ID. ...*/
2091  public function find($id, $columns = ['*']){...}
2095
2096  /** Get a single column's value from the first result of a query. ...*/
2102  public function value($column){...}
2108
2109  /** Execute the query as a "select" statement. ...*/
2115  public function get($columns = ['*']){...}
2121

```

В примере применяются методы **get** и **find**. Они позволяют выполнить SQL-запросы, которые мы писали ранее. Рассмотрим и другие примеры:

- агрегатные функции: **count**, **max**, **min**, **avg** и **sum**. Пример использования:
 - o `DB::table('news')->count();`
 - o `DB::table('news')->max('id');`
- объединение таблиц:
 - o `$users = DB::table('users')->join('contacts', 'users.id', '=', 'contacts.user_id')->join('orders', 'users.id', '=', 'orders.user_id')->select('users.*', 'contacts.phone', 'orders.price')->get();`
 - o `$users = DB::table('users')->leftJoin('posts', 'users.id', '=', 'posts.user_id')->get();`
- условия: **where**, **orWhere**, **whereBetween**, **whereNotBetween**, **whereIn**, **whereNotIn**, **whereNull**, **whereNotNull**, **whereDate**, **whereMonth**, **whereDay**, **whereYear**:
 - o `$users = DB::table('users')->where('votes', '=', 100)->get();`
 - o `$users = DB::table('users')->where(['status', '=', '1'], ['subscribed', '<>', '1'])->get();`
 - o `$users = DB::table('users')->where('votes', '>', 100)->orWhere('name', 'John')->get();`
 - o `$users = DB::table('users')->whereIn('id', [1, 2, 3])->get();`

Практическое задание

1. Продумать структуру сохранения данных следующих таблиц:
 - a. Новости.
 - b. Категории новостей.

с. Источники получения новостных данных.

Структура может быть реализована в виде рисунка, таблиц или специальных диаграмм.

2. Сделать миграции для таблиц из первого задания. Миграции должны иметь ролбеки.
3. С помощью сидинга реализовать заполнение созданных таблиц фейковыми данными. Таблица категорий — минимум 5 записей. Таблица новостей — минимум 10 записей на каждую категорию. Таблица источников — минимум 10 записей.
4. Удалить созданные методы в контроллере для хранения данных и реализовать вывод данных из базы.

Дополнительные материалы

1. <https://laravel.com/docs/5.8/database>.
2. <https://laravel.com/docs/5.8/queries>.
3. <https://laravel.com/docs/5.8/migrations>.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. <https://laravel.com/docs/5.8/>.
2. <http://laravel.su/>.