**Extends Model**

**- Псевдонимы:**

use App\Models\Bbs;

use App\Models\Test as T;

use App\Models\Product as P;

**- Найти все товары в БД:**

//Все методы поддерживают выборку по колонкам в форматах ('columnName1', ...,

'columnNameN') или (['columnName1', ..., 'columnNameN'])

Bb::all(); Bb::get();

$objBb = Bb::select() - так же поддерживает псевдонимы колонок к которым можно

обращатся у объекта(7.5.1).

**- Вернуть определенное количество записей**(7.3.6)**:**

Bb::limit(5)->get(); Bb::take(5)->get();

**- Пропустить определенное количество записей**(применяется только с limit 7.3.6)**:**

Bb::limit(5)->offset(5)->get(); Bb::limit(5)->skip(5)->get();

**- Найти все товар в БД, отсортировать по увеличению цены** (7.3.5)**:**

Bb::orderBy('price')->get(); - есть парраметр asc и desc

**- Найти все товар в БД, отсортировать по убыванию цены:**

Bb::orderByDesc('price')->get();

**- Найти все товар в БД, отсортировать "последнее по дате первое в списке":**

Bb::latest()->get();

**- Выборка уникальных записей** (без повторов 7.3.7)**:**

User::distinct()->get('age');

**- Найти все товар в БД, отсортировать в случайном порядке:**

Bb::inRandomOrder()->get();

Bb::inRandomOrder()->first(); - рандомное одно

**- Найти товар в БД**(7.3.3)**:**

Bb::find(id);

Bb::findMany([id1, id2], ['column name1']);

**- Найти кортеж в БД или сгенерировать исключение** Illuminate\Database\Eloquent\ModelNotFoundException(7.3.2)**:**

Bb::findOrFail();

**- Найти кортеж в БД или создать новый объект модели:**

Bb::findOrNew ();

**- Найти товар в БД, где..:** (7.3.4.)

Bb::where('id', '>=', 10)->get();

Bb::where('title', '=', 'снег')->get(); **=** Bbs::where('title', 'снег')->get();

**- Найти первое совпадение в БД:**

Bb::where('id', '>=', 10)->first();

**- Сравнить две колонки между собой**(7.3.4)**:**

Bb::whereColumn('created\_at', 'updated\_at')->get(); - вернет объявления без изменений

**- Other where**(7.3.4, 7.3.4.1-JASON, 7.3.3; 7.5.4)**:**

orWhereColurnn (), whereDate (), orWhereDate (), whereDay (), orWhereDay (), whereMonth(), orWhereMonth(), whereYear(), orWhereYear(), whereTime(), orWhereTirne(), whereBetween(), whereNotBetween(), orWhereBetween(), orWhereNotBetween(), whereIn(), whereintegerinRaw(), whereNotin(), whereintegerNotinRaw(), orWherein(), orWhereNotin (),

whereNull (), whereNotNull(), orWhereNull(), orWhereNotNull(), firstWhere (); whereExists(), orWhereExists(), whereNotExists(), orWhereNotExists(),

**-Скобочки в SQL запросе с помощью anonymous function**(7.4, 7.3.4)**:**

//Выбрать кортежи где **id больше 80** и возраст бальше 40 или равен 18.

SELECT \* FROM tests WHERE id > 80 AND (age > 40 OR age = 18);

use App\Models\Test as T;

$obj = T::where('id', '>', 80)->

where(function($query){

$query->where('age', '>', '40')->

orWhere('age', 18);})->get();

//Анонимная функция добавляет в SQL запрос скобки. Без скобок 18-ти летних будет искать и до 80-го id.

SELECT \* FROM tests WHERE id > 80 AND age > 40 OR age = 18;

$obj = T::where('id', '>', 80)->where('age', '>', '40')->orWhere('age', 18)->get();

**- Фильтрация записей по значениям полей типа JSON** (7.3.4.1.)**.**

**- Найти первый кортеж в БД:**

Bb::first();

**- Найти первый кортеж в БД удовлетворяющий заданному в параметрах условию**(7.3.3)**:**

Bb::firstWhere ();

**- Найти первый кортеж в БД или возбуждается исключение** Illuminate\Database\Eloquent\ModelNotFoundException(7.3.2)**:**

Bb::firstOrFail();

**- Найти первый кортеж в БД или вернуть анонимную функцию**(7.3.2)**:**

$rubric = Rubric::firstOr(function (){

return new Rubric(['name' => 'Строения']);

});

**- Найти в базе данных и создать объект или не найти и создать объект:** (стр.139)

$objBb = Bb::firstOrNew(['name' => 'Ivan'], ['age' => 18]); - по 1 масс ищет, 1 и 2 вставит

**- Создать пустой объект из класса модели:**

$objBb = new Bb();

**- Занести/изменить данные в поле объекта:**

$objBb->title = 'новое название';

$objBb->user\_id = Auth: :user ()->id; - (не проверенно практически)

**- Занести/изменить данные в поле объекта массовым присвоением:**

protected $fillable = ['title', 'content', 'price'] - поля должны быть указанны в этом свойстве;

$objBb->fill(['title' => 'smile', 'content' => 'some text', 'price' => 100]);

**- Создать пустой объект из класса модели и занести данные в поля массовым присвоением:**

$objBb = new Bb('name' => 'Ivan', 'age' => 18);

**- Сохранить существующий объект в БД:** (так же см. 6.1.1.)

$objBb->save();

$objBb->save(['touch' => true/false]); - заносить/не заносить временные отметки

**- Создать объект при помощи статического метода create() класса модели, занести данные массовым присвоением и сохранить в БД:**

$objBb = Bb::create(['name' => 'Ivan', 'age' => 18]); - метод сохраняет в БД сам

**- Изменить данные существующего объекта и сохранить в БД:**

$objBb->update(['name' => 'Ivan', 'age' => 18]);

**- Найти в БД, изменить и сохранить или не найти, создать и сохранить:** (6.1.2.)

Bb::updateOrCreate(['name' => 'Ivan'],['name' => 'Anton']);

**- Найти в базе данных и создать объект или не найти и создать объект и сохранить:**

$objBb = Bb::firstOrCreate(['name' => 'Ivan'], ['age' => 18]); - по 1 масс ищет, 1 и 2 вставит

**- Задание параметров запросов при выполнении условия**(7.3.8 смотри!!!)**:**

$var = 1;

$objT = T::when($var > 5,

function(){return T::find(1);},

function(){return T::find(10);}

);

**Смена типа выдаваемых значений**(7.3.9)**:**

$bb = Bb::withCasts(['price' => 'int'])->first()->price;

**- Изменен ли объект по сравнению с БД:** (6.1.2.2.)

**- Проверка наличия записей в полученном результате**(7.3.11.)**:**

exists(); doesntExist(); existsor(); doesntExistOr();

**- Объединение результатов от разных запросов**(7.3.12.)**:**

union(); unionAll();

**- Присоединение колонок в одном запросе**(7.5.1.)**:**

select () + addSelect ();

$ objBb = Bb::select('title')->addSelect(['content **as** description']) ->get(); - псевдоним

**- Удалить кортеж из БД через созданный из него объект:**

$objBb->delete();

**- Удалить один или несколько кортежей из БД через метод модели:**

Bb::destroy(id); Bb::destroy(10, 20, 30); Bb::destroy([10, 20, 30]);

**- Восстановление кортежа после мягкого удаления:** (см.разд. 6.1.3.1. и 4.1. 3. 2 и 5. 3. 4)

$objBb->restore();

**- Удалить кортеж окончательно в БД с "Мягким" удалением:**

$objBb->forceDelete()

**- Очистить таблицу( удалить все кортежи, не удаляет если с таблицей есть связи):**

use App\Models\Product as P;

use Illuminate\Support\Facades\DB;

P::truncate()->get();

DB::table('products')->truncate();

**- Параметры полей модели (5.3.1. ):**

**protected $fillable = ['title', 'content', 'price'];** - массив с именами полей, доступных для массового присваивания. Значение по умолчанию - «пустой» массив;

**protected $guarded = ['id', 'created\_at', 'updated\_at'];** - массив с именами полей, недоступных для массового присваивания. Значение по умолчанию- массив с единственным элементом "\*" (все поля не доступны для массового присваивания).

**protected $attributes('price'=>100.0, 'publish'=>true];** - ассоциативный массив со значениями по умолчанию, заносимыми в поля сразу при создании записи. Ключи элементов массива должны соответствовать полям, а значения элементам укажут значения этих полей по умолчанию.

**- Нестандартные параметры обслуживаемой таблицы (5.3.2.):**

Если обслуживаемая таблица не соответствует соглашениям, следует явно указать ее параметры в защищенных свойствах класса модели.

Например не нужны поля CREATED АТ UPDATED АТ

**- Модель может преобразовать тип данных пришедших из БД (5.3.3)**

**- Мягкое удаление, реализация в модели (5.3.4)**

use SoftDeletes; - трейт

**- Найти все + "мягко удаленные" записи**(7.7)**:**

withTrashed().

**- Найти только "мягко удаленные" записи**(7.7)**:**

onlyTrashed().

**- Создание связей между моделями (5.4.):**

**Один-со-многими (5.4.1.)**

hasMany(), belongsTo()

**Один-с-одним (5.4.2.)**

has0ne(), belongsTo()

**- Работа со связанными записями - смотри Cribs One to Many**

**- Пометка записи первичной модели как исправленной при правке или удалении связанных записей вторичной модели (5.4.3.)**

**- Многие-со-многими (5.4.4.)**

belongsToMany(), belongsToMany() + свойство pivot

**- Использование связующих моделей (5.4.4.1.)**

belongsTo(), belongsTo()

**- Сквозная связь один-со-многими (5.4.5.)**

hasManyТhrough()

**- Сквозная связь один-с-одним (5.4.6.)**

hasOneThrough()

**- Записи-заглушки (5.4.7.)**

Если запись вторичной модели не связана ни с одной записью первичной модели

withDefaut()

- **Замкнутая связь (5.4.8.)** (таблица связана сама с собой, данные раздела и подраздела хранятся в одной таблице):

**миграция**

$table->unsignedВiginteger('parent\_id')->nullable();

$table->foreign('parent\_id')->references('id')->on ( 'rubrics') ->onDelete ( 'restrict') ;

**методы модели**

hasMany(self::class, 'parent id'), belongsTo(self::class, 'parent\_id')

**- Объявление метода, реализующего нужную функциональность (5.5)**

**- Преобразование значений полей. Акцессоры и мутаторы (5.6.)**

**-** При **массовом присвоении** значений полей (когда заносится сразу много кортежей в таблицу), **изменении** и **удалении**, автор рекомендует использовать методы построителя запроссов, а не модели. Но в таком случаи проля таблицы которые модель заполняет

втоматически нужно будет заполнять в ручную. (6.2)

**- Вставка фрагментов SQL-кoдa в запрос(7 .5.2.):** например для вызова встроенной функции СУБД

**-------------------------Агрегатные вычисления**(7.6)**-------------------------**

**- Посчитать количество кортежей в таблице модели:**

P::count();

P::count('columnName'); - null не суммирует

P::where('id', '>', 13)->count();

**- Сумма числовых значений столбца в таблице:**

T::sum('age');

**- Фильтрация:**

T::sum('age');

T::where('age', '<', 40)->sum('age');

**- Наименшее значение:**

T::min('age');

**- Наибольшее значение:**

T::max('age');

**- Среднеарифметическое значение по столбцу:**

T::avg('age');

T::average('age');

**-----------------------------------------------------------------------------------------------------------------**

**- Агрегатные вычисления по группам записей**(7.6.2)**:**

groupBy(), having(), orHaving()

**- Join сморти в One to Many.**

**- Использование вложенных запросов** (7.5.4.)**:**

Если необходимо сделать запросс к БД с выборкой на основании какого-то поля таблицы.

**- Установление связи с вложенными запросами**(стр.178)**:**

joinSub(), leftJoinSub(), rightJoinSub()

**- Получение количества связанных записей**(7.6.3)**:**

**У объекта:**

use App\Models\Test as T;

$obj = T::find(11);

$obj->loadCount('products');

В результате у объекта модели появится новое свойство с именем «имяСвязиМоделей\_count» и колличеством связанных записей.

**У модели:**

T::withCount('products')->get();

В результате появится массив объектов модели и у каждого будет свойство с именем

«имяСвязиМоделей\_count» и колличеством связанных записей.

**- Сравнение записей**(7.8)**:**

Метод is модели возвращает true, если текущая и переданная в параметре записи имеют одинаковый ключ, хранятся в одной и той же таблице одной и тойже базы данных, и false - в противном случае:

$rubricl = Rubric::find(9);

$rubric2 = Rubric::firstWhere('name', 'Дачи');

$rubricl->is($rubric2);

**- Получение значения заданного поля**()**:**

**одного поля:**

T::find(11)->value('name');

**всего столбца:**

T::pluck('name');

**- Повторное считывание записей**(7.10)**:**

refresh() - восстанавливает измененные свойства в объекте данными из БД

fresh () - вернет новый объект с данными из БД

**- Принудительная загрузка связанной модели в свойство:**

https://laravel.com/docs/10.x/eloquent-relationships#eager-loading-by-default

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\BelongsTo;

class Book extends Model{

protected **$with** = ['author'];

public function author(){

return $this->belongsTo(Author::class);

}

}