/\*

**Связь «Один-со-Многими»(6.1.4.1.)**

По соглашению именования в Laravel, имя таблицы должно быть во множественном числе(car**s**, engine**s**, primar**ies**, secondar**ies**); наименование связующей колонки во вторичной таблице в единственном числе(ca**r**\_id, primar**y**\_id); наименование метода в первичной модели во множественном числе(engine**s**(), secondar**ies**()), во вторичной модели в единственном числе (ca**r**(), primar**y**()).

**«Один-с-Одним»**

Имя таблицы должно быть во множественном числе(car**s**, engine**s**, primar**ies**, secondar**ies**); наименование связующей колонки во вторичной таблице в единственном числе(ca**r**\_id, primar**y**\_id); наименование метода в первичной модели в единственном числе(engin**e**(), secondar**y**()), во вторичной модели в единственном числе (ca**r**(), primar**y**()).

\*/

**----------CARS AND ENGINES----------**

**Миграция первичной таблицы:**

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

$table->id();

$table->string('model', 50);

$table->timestamps();

});

**Миграция вторичной таблицы:**

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

$table->id();

$table->string('type', 50);

$table->foreignId('car\_id')->constrained()->onDelete('cascade');

$table->timestamps();

);

/\*

foreignId('car\_id') - создаст во вторичной таблице колонку 'car\_id';

constrained() - "создаст из" foreignId('car\_id') связь колонки 'car\_id' вторичной таблицы с колонкой "id" первичной таблицы cars.

\*/

**Модель первичной таблицы:**

use App\Models\Engine;

class Car extends Model{

protected $fillable = ['model']; - чтобы работало массовое присвоенме

public function engines(){

return $this->hasMany(Engine::class);

}

}

**Модель вторичной таблицы:**

use App\Models\Car;

class Engine extends Model{

protected $fillable = ['type'];

public function car(){

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

}

}

/\*

Внимание! Имя метода связи вторичной модели с первичной должно совпадать с именем колонки вторичной таблице по таким правилам: **car**() = **car**\_id и быть в единственном числе. Имя метода связи первичной модели со вторичной должно быть во множественном числе **engines()**.

\*/

**Artisan tinker:**

use App\Models\Engine as E;

use App\Models\Car as C;

//Создать объект первичной модели и сохранить в БД:

$objC = new C();

$objC->model = 'Sexy';

$objC->save();

//Создаем объект первичной модели из первого кортежа первичной таблицы:

$objC = C::find(1);

$objE = new E(); - создали пустой объект вторичной модели

$objE->type = 'V8'; - записали новое значение в поле type

//**Сохраняем запись** во вторичную таблицу связывая ее с первичной таблицей **при помощи вторичной модели** и ее метода связи с первичной моделью primary():

$objE->car()->associate($objC);

$objE->save();

$objE = new E(); - создаем заново пустой объект вторичной модели

$objE->type = 'V12'; - записываем новое значение в поле type

//**Сохраняем запись** во вторичную таблицу связывая ее с первичной таблицей **при помощи первичной модели** и ее метода связи со вторичной моделью engine():

$objC->engines()->save($objE);

**----------PRIMARIES AND SECONDARIES----------**

**Миграция первичной таблицы:**

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

$table->id();

$table->string('name', 50);

$table->timestamps();

});

**Миграция вторичной таблицы:**

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

$table->id();

$table->string('car', 50);

$table->foreignId('primary\_id')->constrained()->onDelete('cascade');

$table->timestamps();

});

/\*

foreignId('primary\_id') - создаст во вторичной таблице колонку 'primary\_id';

constrained() - "создаст из" foreignId('primary\_id') связь колонки 'primary\_id' вторичной таблицы с колонкой "id" первичной таблицы primaries.

\*/

**Модель первичной таблицы:**

class Primary extends Model{

protected $fillable = ['name'];

public function secondaries(){

return $this->hasMany(Secondary::class);

}

}

**Модель вторичной таблицы:**

class Secondary extends Model{

protected $fillable = ['car'];

public function primary(){

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

}

}

Внимание! Имя метода связи вторичной модели с первичной должно совпадать с именем колонки вторичной таблице по таким правилам: **primary**() = **primary**\_id.

**Artisan tinker:**

use App\Models\Primary as P;

use App\Models\Secondary as S;

//Создать объект первичной модели и сохранить в БД:

$objP = new P();

$objP->name = 'Anton';

$objP->save();

//Создаем объект первичной модели из первого кортежа первичной таблицы:

$objP = P::find(1);

$objS = new S(); - создали пустой объект вторичной модели

$objS->car = 'bmw'; - записали новое значение в поле car

//**Сохраняем запись** во вторичную таблицу связывая ее с первичной таблицей **при помощи вторичной модели** и ее метода связи с первичной моделью primary():

$objS->primary()->associate($objP);

$objS->save();

$objS2 = new S(); - создаем еще один пустой объект(второй) вторичной модели

$objS2->car = 'Lotus'; - записываем новое значение в поле car

//**Сохраняем запись** во вторичную таблицу связывая ее с первичной таблицей **при помощи первичной модели** и ее метода связи со вторичной моделью secondary():

$objP->secondaries()->save($objS2);

**----------РАБОТА СО СВЯЗАННЫМИ ЗАПИСЯМИ----------**

**Artisan tinker:**

use App\Models\Car as C;

use App\Models\Engine as E;

**- Метод create():**

//создаем объект нового класса авто и сохраняем его в БД

$objC = C::create(['model' => 'Muscles']);

/\*из объекта первичной модели через метод связи со вторичной моделью сохраняем в БД объект вторичной модели заполнив его поле массовым присвоением\*/

$objС->engines()->create(['type'=>'v8 - twin turbo']);

//или несколько записей

$objC->engines()->createMany([['type'=>'v8 - four turbo'], ['type'=>'v8 - octo turbo']]);

**- Также поддерживает методы:** firstornew (), firstorcreate () и updateorcreate (), описанные в разд. 6.1.1 и 6.1.2.

**- Доступ и изменение свойства связанной записи методами push() и save():**

Доступ к связанным записям объекта можно получить обритившись к его свойству, одноименному с методом связи (вернет одну или коллекцию связанных записей)

$objE->car->model;

$objE->car->model = 'Muscles car';

//метод вторичной модели сохраняющий изменение связанной первичной модели

$objE->push();

//или из вторичного объекта вызвать метод сохранения первичного объекта

$objE->car->save();

$objC->engines[0]->type;

$objC->engines[0]->type = 'v8 - twin turbo the best';

$objC->push();

$objC->engines[0]->save();

foreach ($objC->engine**s** as $engine){

echo $engine->type;

}

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

SELECT \* FROM bbs WHERE rubric\_id = 1 **AND** **(**title = 'ЗИЛ' **OR** price > 4000000**)**

$bbs = $rubric->bbs()->

where(function ($query){

return $query->where('title', 'ЗИП')-> //по идеи можно без return

orWhere('price', '>', 4000000);})->get();

SELECT \* FROM bbs WHERE rubric\_id = 1 **AND** title = 'ЗИП' **OR** price > 4000000

$bbs = $rubric->bbs()->where('title', 'ЗИП')->orWhere('price', '>', 4000000)->get();

//Запрос не верной формы - без скобочек вернет не только из первой рубрики, а и из

остальных.

(7.4) Для выборки связанных записей Laravel вставляет в формируемый запрос SQL-команду WHERE с условием формата <поле внешнего ключа> = <ключ записи первичной таблицы>. Последующий вызов метода where () добавляет следующее условие посредством логического оператора AND. Однако если далее идет вызов метода orWhere (), очередное условие будет добавлено уже посредством оператора OR, имеющего более низкий приоритет исполнения, чем AND. Т.е. используя такой порядок операторов в связанной

модели без скобочек приведет к неправильной выборке.

**- Для удаления записей необходимо воспользоватся методами delete() и destroy()**

**описанными в разделе 6.1.3.**

**- Фильтрация записей первичной таблицы по наличию связанных записей во вторичной таблице**(7.4)**:**

$rubrics = Rubric::whereNotNull('parent\_id')->has('bbs')->get();

Other has(): orHas(), doesntHave(), orDoesntHave(), whereHas(), orWhereHas(),

whereDoesntHave(), orWhereDoesntHave(),

**--------------------Склеивание таблиц в запроссе или SQL Join**(7.5.3.)**--------------------**

Автор книги пишет, что при извлечении данных связанной модели Laravel делает два запроса к СУБД (по первичной и по вторичной моделям). Как альтернативу ускоряющую работу сайта предлогает задать связь между таблицами при формирмвании запроса. По

сути это аналог Join в MySQL.

SELECT \* FROM tests JOIN products ON tests.id = products.test\_id;

use App\Models\Test as T;

**- Первый формат**(7.5.3.)**:**

$obj = T::select()->join('products', 'tests.id', '=', 'products.test\_id')->get();

**- Псывдонимы столбцов таблиц:**

Каждая строка SQL ответа (склеенных таблиц) в Laravel представленна отдельным объектом в массиве. Т.к. имя столбца таблицы становится именем свойства объекта, то из одинаковых столбцов в свойство перейдет только один(при наличии id в нескольких таблицах Laravel id объекта назначит из собственных соображений). Чтобы записать в объект одноименные столбцы им нуждо назначить псевдонимы(или переименовать вообще все).

**- Первый формат с псевдонимами:**

$arrObjs = T::**select**('tests.id **as** test\_id', 'tests.name as test\_name', 'tests.age as test\_age', 'tests.created\_at as test\_created\_at', 'tests.updated\_at as test\_updated\_at', 'products.id as product\_id', 'products.item as product\_item', 'products.created\_at as product\_created\_at', 'products.updated\_at as product\_updated\_at')->**join**('products', 'tests.id', '=', 'products.test\_id')->get();

**- Добавление фильтрациив запроссе:**

$arrObjs = T::select('tests.id as test\_id', 'tests.name as test\_name', 'tests.age as test\_age', 'tests.created\_at as test\_created\_at', 'tests.updated\_at as test\_updated\_at', 'products.id as product\_id', 'products.item as product\_item', 'products.created\_at as product\_created\_at', 'products.updated\_at as product\_updated\_at')->join('products', 'tests.id', '=', 'products.test\_id',)->**where('tests.id', '>', 14)**->get();

**- Второй формат с анонимной функцией:**

$arrObjs = T::**select**()->**join**('products', **function**($join){$**join->on**('tests.id', '=', 'products.test\_id');})->get();

**- Второй формат с анонимной функцией и псевдонимами:**

$arrObjs = T::**select**('tests.id as test\_id', 'tests.name as test\_name', 'tests.age as test\_age', 'tests.created\_at as test\_created\_at', 'tests.updated\_at as test\_updated\_at', 'products.id as product\_id', 'products.item as product\_item', 'products.created\_at as product\_created\_at', 'products.updated\_at as product\_updated\_at')->**join**('products', **function**($join){$**join->on**('tests.id', '=', 'products.test\_id');})->get();

**- Добавление фильтрациив запроссес анонимной функцией :**

$arrObjs = T::select('tests.id as test\_id', 'tests.name as test\_name', 'tests.age as test\_age', 'tests.created\_at as test\_created\_at', 'tests.updated\_at as test\_updated\_at', 'products.id as product\_id', 'products.item as product\_item', 'products.created\_at as product\_created\_at', 'products.updated\_at as product\_updated\_at')->join('products', function($join){$join->on('tests.id', '=', 'products.test\_id')->**where('products.id', '>', 14)**;})->get();

**- Альтернатива where**(стр.173)**:** on(), orOn().

**- Альтернатива join** (стр.173)**:** leftJoin(), rightJoin(), crossJoin().

**- Получить доступ к свойству объекта в массиве:**

$arrObjs[3]-> test\_name;

var\_dump($arrObjs[0]);

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