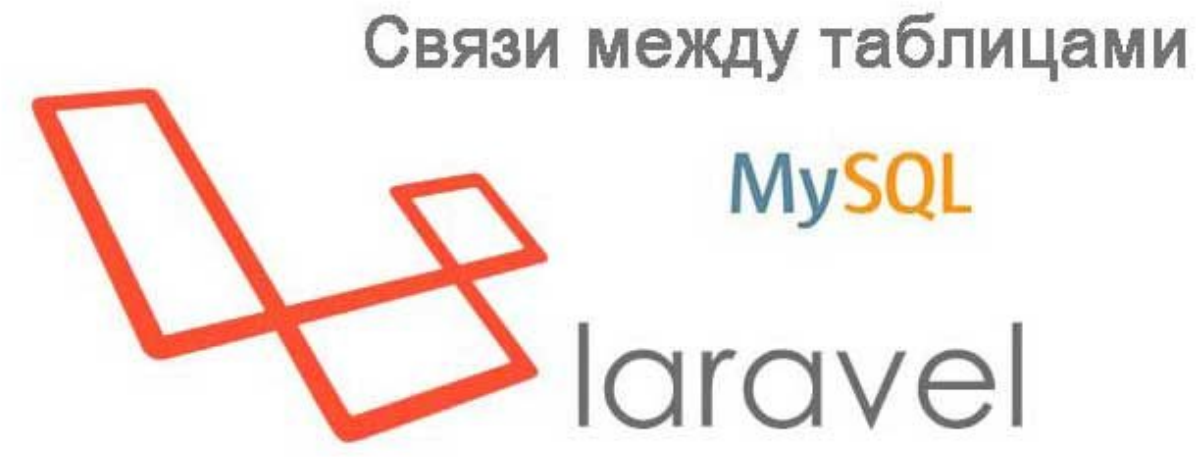
Laravel - створення і використання зв'язків між таблицями БД MySQL.

04.05.2019 Cinex Веб-розробка 0



Зв'язки допомагають нам контролювати таблиці під час роботи з базою даних - не зможете вставити значення, відсутнє в відповідному стовпці пов'язаної батьківської таблиці. Так само,

ПОШУК

ПОШУК …

ПОЗНАЧКИ

APACHE AUTORUN BASH BITCOIN BLOCKCHAIN COMPOSER CRYPTOCURRENCY CSS DATABASE DJANGO DOCKER ELOQUENT ENGLISH ENUM GAMES GIMP GIT HTML JAVA JAVASCRIPT JQUERY LARAVEL MERCURIAL PARSING PHOTOSHOP PHP PHPSTORM

не зможете, випадково, видалити рядки в батьківській таблиці якщо вони використовуються для зв'язку з дочірньою таблицею. Наприклад, якщо клієнт зробив замовлення, то вже не вийде видалити його з бази даних просто так.

При зв'язуванні таблиць ставленням один до одного або один до багатьох, потрібно спочатку визначити яка з них є батьківською - у неї зв'язок робиться по існуючому первинному ключу (зазвичай це поле «id») і як дочірньої - в ній потрібно буде створити додаткове поле для зв'язку з батьківської таблиці, наприклад «user_id» і зовнішній ключ для цього поля.

Так, наприклад, користувач може зробити кілька замовлень, але замовлення не можуть бути без користувача (покупця). тут:

- users – батьківська таблиця;
- customers – дочірня таблиця.

В даному випадку, що зв'язує поле user_id і зовнішній ключ для нього необхідно створювати в дочірній таблиці (customers).
Розглянемо особливості роботи зі зв'язаними таблицями в php-фреймворку Laravel.

Приклад створення файлу міграції в консолі.

Створити порожній файл міграції з назвою CreateRoleTables:

```
1 php artisan make:migration CreateRoleTables
```

Створити файл міграції з назвою CreateRoleTables і визначити в ньому створення таблиці roles:

```
1 php artisan make:migration CreateRoleTables --create=roles
```

PLAYONLINUX PLUGINS PROBLEM SOLVING PYTHON REGEXP SEO SOFTWARE ENGINEERING SPRING SQL TELEGRAM TIPS AND TRICKS TIPS AND TRICKS UNIT TESTING WORDPRESS ВІРШІ ШПАРГАЛКИ

ОСТАННІ НОТАТКИ

20 Laravel Eloquent порад і трюків
15.12.2019

Як зробити скріншот сайту по URL на PHP
04.08.2019

Docker Cheat Sheet
05.07.2019

Гарячі клавіші Ubuntu Linux
01.07.2019

Git happens! 6 типових помилок Git і як їх виправити
22.06.2019

НЕДАВНІ КОМЕНТАРІ

Приклади міграцій для створення зв'язуючого поля/таблиці і зовнішніх ключів.

Міграція-створення зв'язуючого поля із зовнішнім ключем (зв'язок один до одного і один до багатьох).
Міграція для створення таблиці 'countries' зі зв'язуючим полем і зовнішнім ключем:

```php
public function up()
{
    Schema::create('countries', function (Blueprint $table) {
        $table->increments('id');
        $table->string('name');
        //створення поля для зв'язування з таблицею user
        table->integer('user_id')->unsigned()->default(1);
        //створення зовнішнього ключа для поля 'user_id', який пов'язаний з полем id таб
        $table->foreign('user_id')->references('id')->on('users');
        $table->timestamps();
    });
}
public function down()
{
    Schema::dropIfExists('countries');
}
```

Ім'я зовнішнього ключа повинно бути-назва володіющої моделі (батьківської таблиці в однині) плюс _id (тут 'user_id') або вказувати явно в зв'язуючому методі моделі.

Міграція-створення додаткового, що зв'язує поля в існуючу таблицю із зовнішнім ключем.

```
public function up()
{
    Schema::table('posts', function (Blueprint $table) {
        $table->integer('user_id')->unsigned()->default(1);
        $table->foreign('user_id')->references('id')->on('users');
    });
}
```

```php
public function down()
{
    Schema::table('posts', function ($table) {
        $table->dropForeign('posts_user_id_foreign');
        $table->dropColumn('user_id');
    });
}
```

Міграція-створення зв'язуючої таблиці (для зв'язку багато до багатьох).
Приклад зв'язування таблиць **roles** і **users**.
Створюємо таблицю **role_user**. Назва її не випадково-вказуються дві зв'язувані таблиці roles і users через нижнє підкреслення в одиничному числі. Або ж довільна назва і тоді вказується явно в моделі, в зв'язуючому методі **belongsToMany()** в якості другого аргументу.

```php
public function up()
{
    Schema::create('role_user', function (Blueprint $table) {
        $table->increments('id');
        $table->integer('user_id')->unsigned();
        $table->foreign('user_id')->references('id')->on('users');
        $table->integer('role_id')->unsigned();
        $table->foreign('role_id')->references('id')->on('roles');
        $table->timestamps();
    });
}
public function down()
{
    Schema::dropIfExists('role_user');
}
```

Вибірка даних, створення зв'язуючих методів моделей.

Зв'язок "один до одного".

Наприклад, у таблиці **countries** створено зовнішній ключ для зв'язку з таблицею **users**. У такому випадку, таблиця **users** є батьківською і потрібно створити метод **hasOne()** в класі її моделі **User**, а для моделі Country створити метод **belongsTo()**.

Створення методу для моделі User:

```php
public function country()
{
    return $this->hasOne('App\Country');
}
```

Eloquent (реалізація шаблону ActiveRecord в Laravel) вважає, що зовнішній ключ відносини називається по імені моделі. В даному випадку передбачається, що це **user_id**. Якщо ви хочете перекрити стандартне ім'я, передайте другий параметр методу **hasOne()**:

```php
return $this->hasOne('App\Country', 'foreign_key');
```

Отримання зв'язку (в таблиці '**countries**') для користувача з id=1:

```
$user  = User::find(1);
$country = $user->country;
$name = $country->name; //Ukraine
```

викликаємо не метод, а однойменну динамічну властивість.

Створення зворотного зв'язку (для таблиці з якою пов'язували).
Метод моделі:

```php
public function user()
{
    return $this->belongsTo('App\User');
}
```

отримання даних аналогічне:

```
$country  = Country::find(1);
```

```php
$user =    $country->user;
$name = $user->name; //Vova
```

Зв'язок "Один-до-багатьох".

На прикладі зв'язку «один автор – кілька постів». АВТОР (таблиця **users**) - пости (таблиця **posts**). Для цього повинен бути створений зовнішній ключ для таблиці постів. Після цього в моделі **User** можна використовувати метод **hasMany()**, а в моделі **Post** метод **belongsTo()**.
Створення методу для моделі **User**:

```php
public function posts()
{
    return $this->hasMany('App\Post');
}
```

Отримання даних:

```php
$user  = User::find(1);
$posts = $user->posts;
foreach ($posts as $post) {
    //
}
```

викликаємо не метод, а однойменну динамічну властивість.

Створення зворотного зв'язку (для таблиці з якою пов'язували).
Метод моделі **Post**:

```php
public function User()
{
    return $this->belongsTo('App\User');
}
```

отримання даних:

```php
$post = Post::find(3);
$user = $post->user;
echo $user->name;
```

Зв'язок "Багато-до-багатьох".

Пов'язуємо дві таблиці: **roles** і **users** створивши таблицю з назвою **role_user**. При використанні довільного імені зв'язуючої таблиці-назва вказується явно другим аргументом в методі **belongsToMany()**:

```php
return $this->belongsToMany('App\Role', 'user_roles');
```

так само можна перекрити імена ключів за замовчуванням:

```php
return $this->belongsToMany('App\Role', 'user_roles', 'user_id', 'foo_id');
```

Створення методу.
Для моделі **User**:

```php
public function roles() {
    return $this->belongsToMany('App\Role');
}
```

Для моделі **Role**:

```php
public function users(){
    return $this->belongsToMany('App\User');
}
```

Отримання даних:

```php
```

```php
$user = User::find(1);
$roles = $user->roles;
foreach ($roles as $role) {
    //
}
```

як і для інших зв'язків, викликаємо не метод, а однойменну динамічну властивість "roles".Для зворотного зв'язку (іншої моделі), отримання даних проводиться аналогічно.

Перевірка зв'язків при вибірці.

Є можливість відібрати дані з таблиці, тільки ті, які мають зв'язок з іншою, зазначеною таблицею.

Наприклад, потрібно відібрати всіх користувачів, які написали якісь пости (мають зв'язок з таблицею постів – в таблиці posts поле user_id відповідає id користувача):

```php
$users = User::has('posts')->get();
foreach ($users as $user) {
    echo $user->name;
}
```

можна вказати кількість зв'язків, яке має бути:

```php
$users = User::has('posts', '>=', '2')->get();
```

можна додати довільну умову вибірки:

```php
$users = User::whereHas('posts', function($q){
    $q->where('description','like','Опис-%');
})->get();
```

Якщо використовувати не динамічну властивість а метод зв'язку.

В такому випадку отримуємо HasMany Object, для отримання даних з якого потрібно використовувати конструктор запитів:

```php
$user  = User::find(1);
$postsHasMany = $user->post();
$posts = $postsHasMany->where('description','like','Опис-%')->get();
foreach ($posts as $post) {
    //
}
```

тобто можна створити додаткові умови для вибірки.

При використанні можуть виникнути конфлікти якщо в умові використовуються поля з однаковими назвами для різних таблиць. В такому випадку потрібно уточнювати до якої таблиці відноситься поле. Наприклад:

Вставка даних.

Для того, щоб автоматично заповнювалися зв'язуючі поля між таблицями, для вставки даних потрібно використовувати спеціальні методи (аналогічні для будь-яких типів зв'язків):

Один з 2-х способів:

```php
$user = User::find(1);
$user->posts()->create([
    'description' => 'Опис',
    'text' => 'text',
    'user_id' => $user->id
]);
```

або

```php
$user = User::find(1);
```

```php
$post = new Post([
    'description' => 'Опис',
    'text' => 'text',
]);
$user->posts()->save($post);
```

При використанні методу **save()** «масової вставки» може з'явитися помилка

MassAssignmentException in Model.php line 225

це значить, що потрібно дозволити вставку зазначених полів в моделі-метод **$fillable()**.

Зберегти кілька пов'язаних моделей можна так:

```php
$posts = [
    new Post(['description' => 'Опис 1', 'text' => 'text']),
    new Post(['description' => 'Опис 2', 'text' => 'text']),
    new Post(['description' => 'Опис 3', 'text' => 'text']),
];
$user = User::find(1);
$user->posts()->saveMany($posts);
```

Аналогічно і вставка для зв'язку багато-до-багатьох (через зв'язує таблицю).

Приклад.

Потрібно створити користувача з зазначеними даними і присвоїти йому статус адміна (в таблиці roles відповідає id=1):

```php
$user = new User(['name' => 'Masha', 'email' => 'email']);
$role = Role::find(1);
$role->users()->save($user);
```

або

```php
$role = Role::find(1);
$role->users()->create([
    'name' => 'Pasha',
    'email' => 'email2'
]);
```

в даному випадку, створиться вказаний користувач в таблиці **users**, а так само в зв'язуючої таблиці "**role_user**" створиться зв'язуючий запис.

Оновлення даних.

Наприклад потрібно для користувача під id=1 (таблиця users) оновити поле 'text' з зв'язаної таблиці постів (posts)

```php
$user = User::find(1);
$user->posts()->where('id',2)->update(['text'=>'new text']);
```

1. спочатку отримуємо модель потрібного користувача;
2. для даної моделі викликаємо зв'язуючий метод;
3. оскільки один користувач може мати кілька записів, оператором where уточнюємо який саме пост (id=2) потрібно змінити;
4. у методі update() вказуємо поля і їх нові значення.

Якщо не вказати умову (where), то змінені будуть всі рядки пов'язані з даним користувачем.

Джерело

LARAVEL TIPS AND TRICKS

« ПОПЕРЕДНІЙ
Система контролю версій Git: розширена шпаргалка

ДАЛІ »
Стандартні локалізації для Laravel 5

ЗАЛИШТЕ ПЕРШИЙ КОМЕНТАР

Залишити коментар

Вашу адресу електронної пошти не буде опубліковано

Коментувати

Ім'я*

Email *

Збережіть моє ім'я, електронну пошту у цьому веб-переглядачі наступного разу, коли я коментуватиму.

ОПУБЛІКУВАТИ КОМЕНТАР