التعامل مع قواعد البيانات باستخدام
Eloquent ORM

السلام عليكم و رحمة الله و بركاته

في المحاضرة السابقة ذكرنا طريقتين لبناء استعلامات قواعد البيانات في لارافل ، و في هذه المحاضرة سنتعرف على الطريقة الثالثة وهي نظام Eloquent ORM .

ORM إختصار لـ **Object relational mapping** (كائن رسم خرائط العلاقات)

هي تقنية برمجية تتيح الاستعلام و معالجة البيانات من قاعدة البيانات باستخدام نموذج الكائنات object-oriented .

لارافيل يتضمن إطار ORM يسمى بـ Eloquent .

Eloquent ORM هو نظام يستخدم في لارافيل يجعل من تطوير تطبيق الويب أسهل وأسرع .. من خلال التركيز على النهج كائنيّ التوجّه object-oriented بدلاً من كتابة استعلامات SQL عادية .

هذا النظام ينتهج طريقة في التفاعل مع جداول قاعدة البيانات باستخدام الأصناف classes ، حيث أن لكل جدول في قاعدة البيانات صنف class يقابله ، و خصائص هذا الصنف Properties هي حقول الجدول ، كل صنف من هذه الأصناف يدعى بـ Model .

يتم إنشاء النماذج (Models) في جذر المجلد App مباشرةً ، و يمكن تغيير مكانها إلى أي موضع آخر .

في نظام الـ**Eloquent** تنفذ العمليات المختلفة على جداول قاعدة البيانات من تحديد و إدراج و تعديل و حذف عن طريق هذه النماذج Models ، كل كائن ينشئ من هذه الأصناف (Models) يعادل سجل جديد في الجدول المماثل . و هذا النهج في التعامل مع قاعدة البيانات يدعى بنمط **Active Record** .

فمثلاً عندما ترغب في إضافة سجل جديد للجدول users فإنك سننشئ نسخة كائن object instance من النموذج User و الذي يمثل الجدول users .

```
$new_user = new User;
$new_user->name = "name";
$new_user->email = "ex@g.com";
```

و يتم إضافة البيانات الجديدة بعد تنفيذ أمر الحفظ

```
$new_user->save();
```

و تكمن ميزة استخدام Eloquent في إمكانية تمثيل العلاقات بين الجداول و إسترجاع البيانات من عدة جداول بدون الحاجة لكتابة استعلامات معقدة .

إنشاء النماذج Models

لبدء استخدام Eloquent ضمن مشروعك ، عليك أولاً إنشاء نموذج Model لكل جدول في قاعدة البيانات

و أسهل طريقة لإنشاء الـ Model تكون عن طريق الـ artisan باستخدام الأمر

```
php artisan make:model Modelname
```

و يمكن توليد ملف الـMigration للجدول و إنشاء Model معاً باستخدام الأمر التالي

```
php artisan make:model Modelname -m
```

و عند تسمية الـنموذج Model يفضل إتباع القواعد المتعارف عليها . ففي العادة اسم الـModel يبدأ بحرف كبير Upper case ، و كذلك يكون بصيغة المفرد ، بينما يكون اسم الجدول في قاعدة البيانات بصيغة الجمع .

مثال على ذلك :

جدول المستخدمين

سيكون اسم الجدول في قاعدة البيانات **users**

و اسم النموذج Model الذي يمثله هو **User**

بهذه الطريقة سيتعرف نظام Eloquent تلقائياً على النموذج Model و ما يمثله من جدول على قاعدة البيانات .

و في حالة أنك وضعت تسميات لا تتماشى مع هذه القواعد فلن يستطيع نظام Eloquent اكتشاف الجدول الذي يمثله هذا النموذج ، لذا سيتوجب عليك تحديد اسم الجدول بتعريف الخاصية table ضمن النموذج بهذا الشكل:

```
protected $table = 'users_tbl';
```

يفترض نظام Eloquent أن المفتاح الأساسي Primary key للجدول هو بالاسم id ، و بالنوع عدد صحيح integer و تزايدي increment. يمكنك أيضاً تغيير هذه الافتراضيات .

و لتحديد اسم للمفتاح الأساسي نقوم بتعيين الاسم من خلال الخاصية primaryKey

```
protected $primaryKey= 'user_id';
```

و لإلغاء خاصية الترقيم التلقائي للمفتاح الأساسي نعين القيمة false للخاصية incrementing

```
public $incrementing = false;
```

أيضاً ينشئ نظام Eloquent قيم على الحقول created_at و updated_at عند كل عملية إضافة أو تعديل ، و إذا رغبت في إيقاف هذا ، قم بتعيين القيمة false للخاصية timestamp

```
public $timestamps = false;
```

كل هذه الخصائص ستحدد من خلال النموذج Model الذي يمثل الجدول المطلوب .

مثال عملي :

لنقم الآن بعمل مثال بسيط لتوضيح طريقة إنشاء النماذج Model . سأقوم بإنشاء جدول في قاعدة البيانات بالاسم **posts** .

لإنشاء الجدول سنحتاج أولاً لإنشاء ملف Migration ، و من الرائع أنه يمكن إنشائهما معاً بأمر واحد فقط ، و لاحظ هنا أننا سنضع اسم النموذج بصيغة المفرد و كذلك سيُكتب أول حرف كـ Capital letter

```
php artisan make:model Post -m
```

ستلاحظ الآن ظهور النموذج Post على الدليل app ، و أيضاً ملف Migration ، بالنظر في هذا الملف ستجد اسم الجدول تم تسجيله تلقائياً بنفس اسم الـmodel و لكن في صيغة جمع

إلى الآن لا يوجد جدول في قاعدة البيانات ، فلنبدأ بإكمال مخطط الجدول في ملف Migration ، سأضيف حقلين بالأسماء

title و سيكون نوعه string

و **body** و نوعه text .

و سأقوم بتغيير اسم الحقل الرئيسي من id إلى post_id

فلنقم بتنفيذ الأمر Migrate لتنشئ الجداول في قاعدة البيانات . **php artisan migrate**

سننتقل للنموذج Post . و كما ذكرنا سابقاً فإن النموذج سيرتبط تلقائياً مع الجدول الذي يطابق اسمه في صيغة جمع ، و هو هنا الجدول posts .

في الجدول posts الذي أُنشيء على قاعدة البيانات هناك الحقل الأساسي بالاسم post_id

بينما النموذج هنا يفترض أن الحقل الأساسي هو id ، إذاً علينا تعيين الاسم الفعلي للحقل بوضع الخاصية

```
protected $primaryKey= 'post_id';
```

الآن الجدول posts جاهز في قاعدة البيانات و كذلك النموذج الذي يمثله ، فلنبدأ بتطبيق استعلامات على هذا الجدول باستخدام نظام Eloquent ، لنصل إلى صفحة مشابهة لما هي أمامك ، بحيث نستطيع من خلالها تطبيق عمليات إضافة و تعديل و حذف البيانات .

تنفيذ عمليات CRUD باستخدام Eloquent :

سنقوم بتنفيذ عمليات CRUD على الجدول posts باستخدام نظام Eloquent .
أنشئت view بالاسم **index** ← ستعرض عليه جميع البيانات الموجودة على الجدول posts .
و سننشئ أيضاً **Controller** ← حيث سيمثل الوسيط بين النموذج Post و الـview المسمى index .

سنقوم بإنشاء الـController عن طريق موجه الأوامر بالاسم **PostController** ، و بما أنه سيخصص لتنفيذ العمليات إدخال و عرض و تعديل و حذف فالأنسب أن يكون من النوع Resource controller

```
$ php artisan make:controller PostController --resource
```

في المتحكم الناتج سنضيف جميع الاستعلامات التي نحتاجها لتنفيذ العمليات السابقة . التعامل سيكون مع النموذج Post إذاً علينا أولاً استدعاءه .

```
use App\Post;
```

الآن نستطيع استخدام جميع المزايا التي يقدمها نظام eloquent

بداخل الدالة index() سأدرج الكود الخاص بجلب جميع البيانات من جدول posts .
سيكتب اسم الكلاس الذي سنتعامل معه و هو هنا Post ثم تستدعى الدالة المطلوبة بطريقة static ، فمثلاً إذا رغبت في جلب جميع البيانات استخدم all() مباشرةً

```
        $posts=Post::all();
```

يمكنك أيضاً استخدام جميع دوال الـquery builder مع eloquent بنفس الطريقة . فمثلاً تستطيع استخدام الدالة get() بنفس الأسلوب المستخدم في query builder ، و ستعطي نفس النتيجة .

```
        $posts=Post::get();
```

البيانات المسترجعة من استعلامات eloquent تكون دائماً بشكل مجموعة من الكائنات collection
و يمكنك التعرف على الدوال المستخدمة معها من هذا الرابط
https://laravel.com/docs/5.5/eloquent-collections#available-methods

يمكنك جلب البيانات حسب شروط معينة ، فمثلاً لجلب جميع البيانات من الجدول حسب ترتيب حقل معين تنازلياً ، سنستدعي الدالة orderBy حيث تستقبل وسيطين ، اسم الحقل الذي سترتب النتائج بناءً عليه ، و نوع الترتيب .
و هو هنا تنازلي (descending) .

```
$posts=Post::orderBy('post_id', 'desc')
        ->get();
```

هذا سيعطينا جميع البيانات في الجدول مرتبة تنازلياً ، لكن ماذا إذا أردنا أول ١٠ صفوف فقط

سنضيف الدالة take() بهذه الطريقة ، حيث الرقم ١٠ هنا يعبر عن عدد الصفوف المطلوب جلبها .

```
$posts=Post::orderBy('post_id','desc')
        ->take(10)
        ->get();
```

و لجلب بيانات حسب شروط معينة ، مثلاً جلب بيانات المشاركة رقم ٩ ، سنستخدم where بحيث يحدد بارامترين .

اسم الحقل و القيمة .

```
$posts=Post::where('post_id', 9)
        ->get();
```

و يمكن استرجاع بيانات بالبحث في حقل المفتاح الأساسي بطريقة أكثر اختصاراً ، باستخدام الدالة find() ، تمرر لهذه الدالة قيمة المفتاح الأساسي لتقوم بالبحث عنها و إرجاع النتيجة في سجل واحد فقط .

```
$posts=Post::find(1);
```

بالطبع هنا يفترض eloquent أن المفتاح الأساسي في الجدول هو id ، مالم تقم بإلغائها override من خلال الخاصية `primaryKey`.

في حالة عدم وجود نتائج مطابقة فإن الدالة `find` ستعيد القيمة null .

هناك دالة أخرى مشابهة هي `findOrFail()` تعمل نفس عمل الدالة find غير أنها ستقوم بعمل استثناء في حالة عدم وجود نتائج و إظهار صفحة خطأ .

لنكمل المثال و سنسترجع جميع البيانات الموجودة في الجدول posts بدون تحديد أي قيود .

```
$posts=Post::all();
```

و الآن سنمرر هذه البيانات إلى الـview المسمى index

```
return view('index',compact('posts'));
```

سنستخدم حلقة **foreach** للمرور على جميع كائنات المصفوفة و طباعتها بشكل جدولي ، و كما تلاحظ فقد قمنا بعرض الثلاثة حقول الموجودة في الجدول posts **(post_id , title , body) .**

```
@foreach ($posts as $rs)
 <tr>
     <td>{{ $rs->post_id}}</td>
     <td>{{ $rs->title}}</td>
     <td>{{ $rs->body}}</td>
 </tr>
 @endforeach
```

لعرض النتيجة على المستعرض ، علينا إنشاء مسار ليقوم باستدعاء الدالة index ، في ملف web.php سننشئ مسار للدالة ، و الأنسب أن نستخدم route::resource لينشئ مسارات لجميع الدوال في PostController ، سأعطي هذا المسار العنوان **posts**

```
Route::resource('posts', 'PostController');
```

كما تعلمت سابقاً فهذا السطر سينتج سبع توجيهات كما ترى هنا .

```
| GET|HEAD  | posts              | posts.index   | App\Http\Controllers\PostController@index   |
| POST      | posts              | posts.store   | App\Http\Controllers\PostController@store   |
| GET|HEAD  | posts/create       | posts.create  | App\Http\Controllers\PostController@create  |
| GET|HEAD  | posts/{post}       | posts.show    | App\Http\Controllers\PostController@show    |
| PUT|PATCH | posts/{post}       | posts.update  | App\Http\Controllers\PostController@update  |
| DELETE    | posts/{post}       | posts.destroy | App\Http\Controllers\PostController@destroy |
| GET|HEAD  | posts/{post}/edit  | posts.edit    | App\Http\Controllers\PostController@edit    |
+-----------+--------------------+---------------+---------------------------------------------+
```

دالة index سيتم تنفيذها عند طلب العنوان posts بنوع الطلب GET ، و posts.index هو اسم لهذا المسار يمكنك أيضاً استخدامه لطلب الصفحة بنفس الطريقة التي ذُكرت في المحاضرة السابقة .

سأقوم بطلب العنوان posts ، لاحظ انه توجه مباشرة لتنفيذ التعليمات البرمجية ضمن الدالة index .

حفظ البيانات

سنرى الآن طريقة حفظ بيانات للجدول ، و سنحتاج واجهة form ليتم من خلاله إضافة المدخلات

أنشئ view جديد و أعطه اسم مناسب و ليكن **create** ، و بداخل هذا الـview ضع كود html للواجهة form التي ستستخدمها لإدخال البيانات .

الزر الموجود على الصفحة **index** (create new post) عبارة عن رابط تشعبي ستكون مهمته عرض الصفحة create .

لنعد لـPostController . و نقوم بإضافة الأكواد المطلوبة لحفظ البيانات .

قد تحتار بين الدالتين create و store حول أي منهما ستضع الكود الخاص بالحفظ لقاعدة البيانات

الدالة **create** تستخدم لعرض الـ form الخاص بإضافة البيانات (و في مثالنا اسم المسار الذي يستدعيها هو posts.create)

أما الدالة **store** هي المسؤولة عن حفظ البيانات (و هنا اسم المسار الخاص بها هو posts.store)

ضمن الدالة create سنرجع ملف العرض create

```
    public function create()
    {
        return view('create');
    }
```

نريد تنفيذ هذه الدالة عند النقر على الرابط create new post ، بالتأكيد تعرف ما الذي ستضيفه هنا ..

اسم التوجيه الذي يستدعي الدالة create هو posts.create إذاً سنضع التالي

```
<a href="{{route('posts.create')}}" class="btn btn-primary">create new post</a>
```

تم عرض الـفورم .

الآن علينا إضافة الكود اللازم لحفظ البيانات .. سيتم إرسال البيانات للقاعدة عند النقر على الزر save و ركز هنا على أن نوع الطلب سيكون post .

في الدالة store() سنضيف كود حفظ البيانات باستخدام eloquent

لإضافة سجل جديد في الجدول Post ، سنقوم أولاً بإنشاء كائن جديد من الكلاس Post

```
        $post=new Post;
```

نعين القيم لكل خاصية (حقل)

```
        $post->title = $request->title_inpt;
        $post->body = $request->body_inpt;
```

البيانات القادمة من صفحة إدخال البيانات و الموجودة على title_input و body_input ستعين كقيم لخصائص الكائن المُنشَأ .

و أخيراً نستدعي الدالة **save()** ليضاف السجل الجديد إلى قاعدة البيانات

```
        $post->save();
```

طريقة أخرى لحفظ البيانات بسطر واحد فقط ، و هي ميزة أضيفت في الإصدار 5.5 من لارافل

و تكون بتمرير جميع قيم requests للدالة create بالشكل التالي

```
        Post::create($request->all());
```

قبل استخدامك لهذه الطريقة تأكد أولاً أن تحدد الحقول التي سيتم تعبئتها ، و سيتم هذا عن طريق كتابة التالي في داخل الـModel الذي يمثل الجدول

```
    protected $fillable =['title','body'];
```

و هنا حددنا أننا سنقوم بتعبئة الحقلين title و body .

أيضاً تأكد من أن أسماء بارامترات الطلب تطابق أسماء حقول قاعدة البيانات ، هذا يعني أننا سنغير التسميات في **صفحة إدخال البيانات** من title_input و body_input إلى title و body .

لنتأكد الآن من عملية الحفظ .. تم الحفظ بنجاح .

تعديل البيانات

لنقم الآن بإضافة رابط هنا لتعديل البيانات لسجل محدد من خلاله ، عند النقر على هذا الرابط سيتم عرض بيانات الصف المحدد على صفحة أخرى ، سأنشئ هذه الصفحة و سأعطيها الاسم edit ، سأضع الشيفرة الخاصة بالـ form الذي ستظهر بداخله البيانات .

هنا (على PostController) سنعمل على الدالتين Edit و Update بخطوات مشابهه لعملية حفظ البيانات .

الدالة **edit()** ستعرض الفيو edit و تستقبل رقم السجل الذي اختاره المستخدم

و الدالة **update()** ستستدعى عند النقر على الزر تحديث ليتم تحديث البيانات فعلياً على قاعدة البيانات

أظن أنك الآن تعرف ما الخطوات التي ستتبعها

أولاً لنفعِّل الرابط Edit بحيث ينقلنا لـview المسمى Edit

{{ Route('posts.edit') }}

هذا العنوان سيوجه إلى الدالة edit() الموجودة ضمن **PostController** ، و لكن **مهلاً** فهذه الدالة تستقبل قيمة و هي معرف السجل ليتم تحديث السجل بناءً عليه ، و في هذا المثال المعرف هو post_id

مرر البيانات المطلوبة عبر الدالة route() بالشكل التالي

```
<a href="{{route('posts.edit',$posts->post_id) }}"> Edit </a>
```

الأمور الآن تعمل بشكل جيد .. فهو يستدعي الدالة edit() و يمرر القيمة كما ترى .

في داخل الدالة Edit() سنجلب بيانات السجل المطلوب تحديثه ، و هو السجل الذي قيمة المفتاح الأساسي فيه مساوية لـ $id ، ثم سنمررها لـview المسمى Edit

```
$post=Post::find($id);
 return view('edit',compact('post'));
```

هذا واضح جداً ، فبعد أن حصلنا على رقم السجل الذي طلب المستخدم تحديثه ، استخدمنا الدالة find() لإيجاد بقية بيانات السجل ثم تمريرها إلى الـview الخاص بالتعديل .

لننتقل للفيو edit سنقوم بإظهار البيانات القادمة بداخل عناصر الفورم ، و بالطبع البيانات موجودة في $post كسجل واحد فقط ، قيم الحقول هنا ستكون بالشكل **$post->title** و **$post->body**

لنشاهد النتائج ..

جيد .. و لكن حتى الآن لم ننتهي .. فبعد وضع القيم الجديدة و النقر على الزر update يتم تحديث قيمة السجل في قاعدة البيانات ، الأمر الآن يشبه ما فعلناه في عملية الحفظ إلى حد كبير ، سنقوم بإرسال القيم الجديدة إلى الدالة update ، عند النقر على الزر بداخل الفورم.

بداخل الـview المسمى edit

ضمن action ضع عنوان المسار الذي سيعمل على التوجيه للدالة update() و هو post.update

(و إذا كنت تشعر بحيرة بخصوص هذه المسارات ، عد قليلاً بذاكرتك إلى موضوع Route resource **)**

لاحظ هنا أنه للانتقال إلى الدالة update فعليك استخدام الاسم posts.update .

الدالة update تستقبل الطلب requests الذي يحوي البيانات الجديدة ، و أيضاً رقم السجل ، إذاً سنمرر بيانات $post_id بالشكل التالي

```
<form action="{{ route('posts.update',$post->post_id) }}" method="post">
```

في الدالة update سنقوم بثلاث خطوات :

الانتقال للسجل المطلوب تحديثه .. استبدال البيانات القديمة بالجديدة .. حفظ التعديل .

```
public function update(Request $request, $id)
{
    $post=Post::find($id);
    $post->title = $request->title;
    $post->body = $request->body;
    $post->save();
}
```

يبدو كل شيء على ما يرام .. و لكن سيظهر لنا أن هناك خطأ ما .. لنعد قليلاً للتأكد من المسار الخاص بالدالة update

```
PUT|PATCH | posts/{post} | posts.update |App\Http\Controllers\PostController@update
```

لاحظ أنه للوصول لهذه الدالة و تنفيذها علينا استخدام اسم المسار posts.update و لكن مع نوع الطلب PUT أو PATCH بينما هنا نوع الطلب هو post ..

المتصفحات لا تدعم سوى أنواع الطلبات GET و POST أي أننا لا نستطيع استخدام PUT مباشرةً ضمن الدالة method ، هناك حل جيد هو إرسال الطلب PUT ضمن حقل مخفي بهذه الطريقة

```
<input name="_method" type="hidden" value="PUT">
```

فلنختبر النتائج على المتصفح .. تمت عملية التعديل بنجاح ..

قم بتطبيق هذا المثال مع إعادة التوجيه إلى الصفحة index بعد كل عملية تعديل

أضف أيضاً رابط لحذف السجل المحدد بطريقة مشابهة لما سبق .

```
<td>
    <form method="post"  action="{{ route('posts.destroy',$rs->post_id) }}">
          {{ csrf_field() }}
          <input name="_method" type="hidden" value="DELETE">
          <input type="submit"  value="delete" class="btn-xs btn-danger">
    </form>
</td>
```

سيتم الحذف باستخدام الدالة delete() بالطريقة الموضحة هنا :

```
$post=Post::find($id);
      $post->delete();
```

العلاقات Relationship

كما تعلم فإن البيانات في قاعدة البيانات تخزن في عدة جداول منفصلة لتفادي التكرار ، و بالتالي نحتاج إلى وسيلة لربط الجداول ببعضها للتمكن من استرجاع صفوف مترابطة من عدة جداول بسهولة و كفاءة .
مثال على ذلك جدول المقالات **articles** و جدول التعليقات **comments** ستحتاج حتماً لاسترجاع صفوف معينة و ما يتعلق بها من الجدول المرتبط ، كالوصول لكافة التعليقات التي تخص مقال معين .
نظام Eloquent يقدم طرق واضحة لتعريف العلاقات بين الجداول ، و بعد تعريف تلك العلاقات يمكن بعدها الاستعلام و استرجاع البيانات من عدة جداول بسهولة كبيرة كما سنرى تالياً .

تختلف أنواع العلاقات بين الجداول و سنذكر الثلاثة أنواع الرئيسية :

- علاقة واحد إلى واحد.(One-to-One)
- علاقة واحد إلى كثير .(One-to-Many)
- علاقة كثير إلى كثير.(Many-to-Many)

❖ **تعريف العلاقات**
تعرف العلاقات بين الجداول بداخل دوال functions ، يتم إضافتها بداخل النماذج Models .

❖ **علاقة واحد إلى واحد.(One-to-One)**

في هذا النوع يرتبط كل سجل من الجدول الأول بسجل واحد من الجدول الثاني ، و كذلك يرتبط كل سجل في الجدول الثاني بسجل واحد فقط من الجدول الأول

مثال على ذلك : جدول المستخدمين users و جدول العناوين Addresses ، كل مستخدم في الجدول users سيمتلك عنوان واحد فقط **(hasOne)** ،
ستعرف العلاقة من User إلى address بداخل النموذج User
ننشئ function يعبر عن الجدول المقابل address ، و نصف بداخلها العلاقة بالطريقة hasOne

```php
class User extends Model
{
    public function address()
    {
        return $this->hasOne('App\Address');
    }
}
```

الدالة hasOne تستقبل اسم النموذج للجدول المقابل ، هذا يعبر على أن كل user سيمتلك عنوان واحد ، لاحظ مدى مقروئية الكود هنا .

يحدد Eloquent المفتاح الأجنبي بناءً على اسم النموذج Model ، ففي المثال السابق سيفترض أن جدول العناوين يشمل المفتاح الأجنبي user_id ، و في حالة لم يكن بهذا الاسم فعليك تحديد اسمه بتمريره كوسيطة ثانية ضمن الدالة hasOne بالشكل :

```
return $this->hasOne('App\Address, 'userid');
```

الآن نستطيع الوصول إلى العنوان المرتبط بمستخدم معين

```
$address= User::find(1)->address->city;
```

هذا يجلب اسم المدينة من جدول address للمستخدم الذي رقمه هو ١ . و هنا جلبنا عنوان المستخدم بمعرفة رقم مستخدم معين .

ماذا إذا أردنا الوصول لمستخدم **user** بمعرفة عنوان معين

```
$user= Address::find(2)->user;
```

سيتحتَّم عليك أولاً تعريف العلاقة من جهة الجدول Addresses

كل عنوان في الجدول addresses ينتمي إلى مستخدم واحد في الجدول users **(belongsTo)**

ضمن النموذج Address سنضيف التالي

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

نستطيع الآن الوصول لبيانات المستخدم بمعرفة عنوان معين .

```
$user= Address::find(2)->user;
```

- **علاقة واحد إلى كثير .(One-to-Many)**

في هذا النوع السجل في الجدول الأول سيحوي من السجلات في الجدول الثاني

و كل سجل في الجدول الثاني ينتمي إلى سجل واحد فقط من الجدول الأول

هذا النوع من علاقات قواعد البيانات هو الأكثر شيوعاً .

مثال على ذلك جدول التدوينات posts و جدول التعليقات comments ،

كل موضوع سيرتبط به العديد من التعليقات hasMany

بينما كل تعليق ينتمي إلى موضوع واحد فقط . `belongsTo`

فلنمثل هذه العلاقات بشكل عملي الآن ، لدي الجدول posts و الجدول comments و لكل منهما الـmodel الخاص به ، سأقوم الآن بتعريف العلاقات بداخل كل نموذج حتى نستطيع الاستعلام بسهولة .

في داخل النموذج Post سنعرف العلاقة بالشكل التالي

```
class Post extends Model
{
    public function comments()
    {
        return $this->hasMany('App\Comment');
    }

}
```

حيث كل موضوع سيمتلك عدد من التعليقات .
الآن نستطيع الوصول لكل التعليقات المرتبطة بموضوع معين بالشكل التالي

```
$comments = Post::find(1)->comments;
```

فلنقم بعرض التعليقات الناتجة على ملف العرض باستخدام حلقة foreach

```
foreach ($comments as $comment) {
   <div>
      <h3>{{$comment->title}}</h3>
      <p> {{$comment->body}} </p>
   </div>
}
```

و إذا أردت الوصول لموضوع معين من خلال التعليق الخاص بها ، عليك تمثيل معكوس العلاقة من خلال تعريف نوع العلاقة في نموذج التعليقات comments

هذا النوع سيعرف بالطريقة `belongsTo` ، حيث أن كل سجل في جدول comments ينتمي إلى سجل واحد من جدول posts .

```
class Comment extends Model
{
    public function post()
    {
        return $this->belongsTo('App\Post');
    }
}
```

لنقم الآن بطباعة بيانات المنشور post المرتبط بتعليق Comment معين

```
$comment = App\Comment::find(1);

echo $comment->post->title;
echo $comment->post->body;
```

❖ **علاقة كثير إلى كثير (Many-to-Many)**

في هذا النوع يرتبط السجل في الجدول الأول بعدة سجلات من الجدول الثاني . و العكس أيضاً حيث يرتبط السجل من الجدول الثاني بعدة سجلات من الجدول الأول .

مثال على ذلك جدول المستخدمين users و جدول الأدوار roles ، كل مستخدم user يمكن أن يسند له أكثر من دور ، و كل دور Role سيشترك فيه العديد من المستخدمين .
هذا النوع من العلاقات يتطلب وجود جدول ثالث للربط ، سيشمل هذا الجدول المفتاح الأساسي لكلا الجدولين .
تسمية جدول الربط يكون بحسب الترتيب الأبجدي لأسماء النماذج Models ،
فمثلاً سيكون جدول الربط للجدولين **users** و **roles** هو role_user .

ستعرف العلاقة من الجهتين بالطريقة **belongsToMany** حيث كل سجل في الجدول الأول سيرتبط بالعديد من السجلات في الجدول الثاني و العكس كذلك

للتمكن من استرجاع الأدوار الخاصة بمستخدم معين ، أو استرجاع جميع المستخدمين المشتركين في نفس الدور سنقوم بوصف العلاقة بداخل النموذج User بالشكل الموضح أمامك

```
class User extends Authenticatable
{
    public function roles()
    {
        return $this->belongsToMany('App\Role');
    }
}
```

الآن نستطيع الوصول لبيانات الأدوار المرتبطة بمستخدم معين

```
        $roles= User::find(2)->roles()->get();
```

و إذا أردت الوصول للمستخدمين المشتركين في دور Role معين ، فعليك أولاً تعريف العلاقة belongsToMany بداخل النموذج Role

```
class Role extends Model
{
    public function users()
    {
        return $this->belongsToMany('App\User');
    }
}
```

الآن يمكنك استرجاع بيانات المستخدمين حسب role معين

```
        $users= Role::find(2)->users()->get();
```