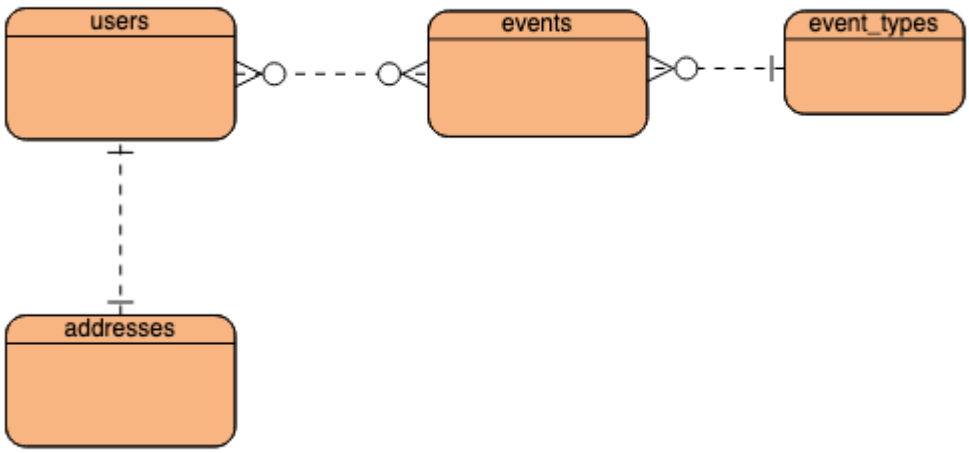
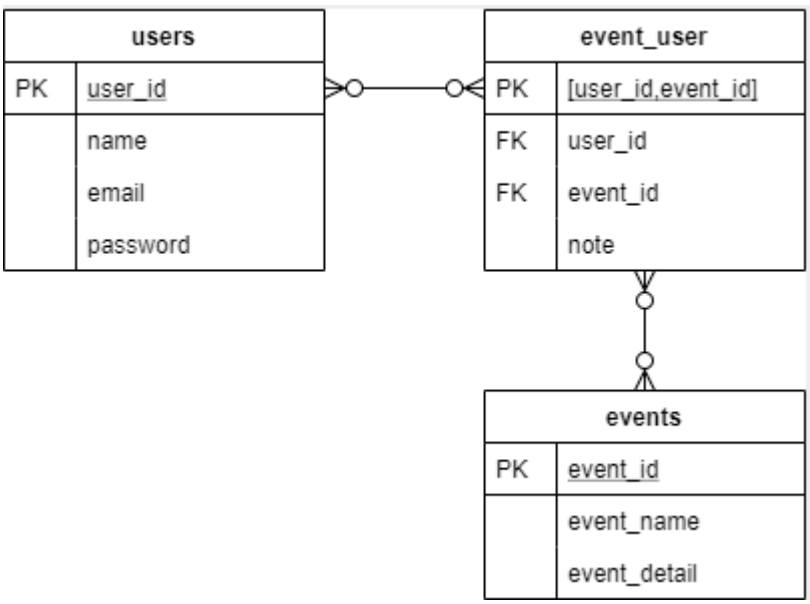
Inés Larrañaga Fernández de Pinedo	CSC Jesuitas - Logroño
	DWES

Table of Contents

ManyToMany relationship



From the previous schema, we need to implement the ManyToMany relationship like this:



Seeing the picture, we can say that one user can attend to many events and that an event can be attended by many users.

In order to make the implementation you will use a pivot table table called `event_user`, with two FK: `user_id` and `event_id`.

Database Migration

We need to create the new `events` and `event_user` table. In order to do so:

```
php artisan make:migration create_events_table
php artisan make:migration create_event_user_table
```

If you're wondering why the table *event_user* name is weird, it's because I used the Laravel naming convention when creating a pivot table name. The naming convention for a pivot table is: put together the two table names in a singular form in alphabetical order. This naming convention will make it easier for us to set up the model later.

Let's open the database/migrations/create_events_table.php.

```
    public function up()
    {
        Schema::create('events', function (Blueprint $table) {
            $table->id();
            $table->string('event_name');
            $table->string('event_detail')->nullable();
            $table->timestamps();
        });
    }
```

Launch the migration tool:

```
php artisan migrate
```

Now, open the database/migrations/create_event_user_table.php.

```
    public function up()
    {
        Schema::create('event_user', function (Blueprint $table) {
            $table->primary(['user_id','event_id']);
            $table->bigInteger('user_id')->unsigned();
            $table->bigInteger('event_id')->unsigned();
            $table->string('note');
            $table->timestamps();
            $table->foreign('user_id')
                ->references('id')
                ->on('users');
             $table->foreign('event_id')
                ->references('id')
                ->on('events');
        });
    }
```

Nothing special here except, the primary key for event_user table. I use `$table->primary(['user_id','event_id'])` it means that, the combination between user_id and event_id must be unique.

```
php artisan migrate
```

In case you have an error, check your ´.env´ file: "DB_HOST=127.0.0.1".

Check your new tables using DataGrip.

Eloquent ORM Model

So now, even if we have 3 tables in our database, we only need 2 models in this project. In this case, we don't need to create a model for `event_user` table because Laravel will store the many-to-many relationships automatically in our pivot table. Let's create the Event model:

```
php artisan make:model Event
```

Then open the app/Event model:

```
class Event extends Model
{
    use HasFactory;
    protected $fillable = [
        'event_name', 'event_detail',
    ];

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

For reading and storing the relationship between users table, we use `belongsToMany()` function without specifying the pivot table name because we are following the naming convention for pivot table. If you have a pivot table without following the naming convention, then you must specify the pivot table name inside the second argument of *belogsToMany()*.

Now, complete the `User` model and add this:

```
    public function events()
    {
        return $this->belongsToMany('App\Models\Event')->withPivot('note')-
>withTimestamps();
    }
```

Controller methods

One of the methods we are going to add in our `UserController` file is the `bookEvent` method, so that we can book an event for an specific user:

```
    public function bookEvent(Request $request, User $user, Event $event)
    {
        $note = '';
        if($request->note){
            $note = $request->note;
        }
        if($user->events()->save($event, array('note' => $note))){
            return response()->json(['message'=>'User Event
Created','data'=>$event],200);
        }
        return response()->json(['message'=>'Error','data'=>null],400);
    }
```

When we are storing the relationship, it's very easy with Eloquent ORM Laravel. We can just call $user->events()->save(event, array('note' => $note))$ function without specifying anything about our pivot table.

In addition, we add another column named *note* in the pivot table and that's the way on how storing this *note* column into our pivot table.

Another interesting method could be to obtain the list of events for an specific user:

```
    public function listEvents(User $user)
    {
        $events = $user->events;
        return response()->json(['message'=>null,'data'=>$events],200);
    }
```

Now, let´s create a new controller called `EvenController`:

```
php artisan make:controller EventController -r
```

Inside, we are going to create a method called `listUsers` that will return all the users that will attend for an specific event:

```
    public function listUsers(Event $event)
    {
        $users = $event->users;
        return response()->json(['message'=>null,'data'=>$users],200);
    }
```

API routing

```
//UsersController routing
    Route::post('users/{user}/events/{event}/book', 'bookEvent');
    Route::get('user/{user}/events', 'listEvents');

//EventsController routing
Route::controller(EventController::class)->group(function() {
    Route::post('event', 'store');
    Route::get('event/{event}/users', 'listUsers');
});
```

Now test the new methods properly.