**Namesco Challenge**

I used Model factory and database Seeder in order to make the design work.

Model factory is a representation of a fake data to get created for a specific model.

This is an example of generating a seeder for film Model

public function run()

    {

        factory(\App\Film::class,1)->create()->each(function ($f) {

            $f->employees()->saveMany(factory(\App\Employee::class,rand(10,20))->make())->each(function ($e) {

                $e->contracts()->saveMany(factory(\App\Contract::class, 1)->make())->each(function ($c) {

                    $c->contract\_details()->saveMany(factory(\App\ContractDetail::class, rand(1,5))->make());

                   });

              });

            });

    }

For each film I create a random number between 10 and 20 of employees, and for each employee I create one contract and for each contract I create a random number between 1 and 5 of contract\_detail which represent different role(s) played by each employee.

I calculate the month ‘s pay via 2 triggers:

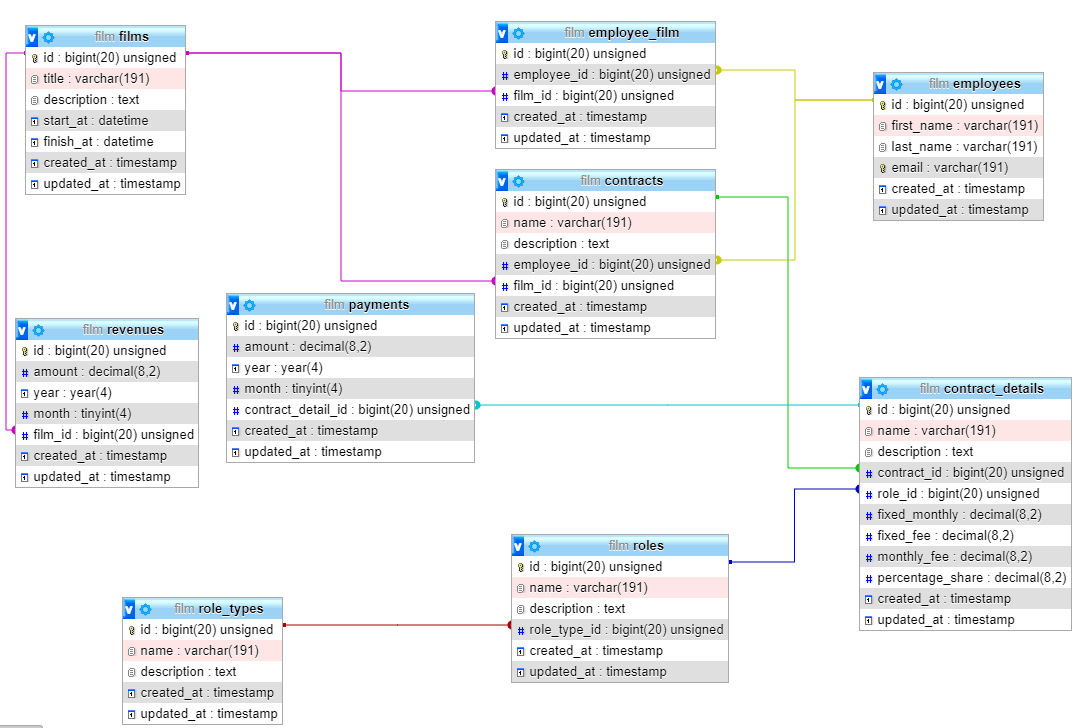
1.One after insert into table contract\_details

2.And one after insert into revenues table

Bouth

**1. Create an entity relationship diagram to represent the database tables and fields that could be used**

**to store this information.**



**2. Create a separate entity relationship diagram to represent the classes and method signatures you**

**would use to model and serve this data.**

class Film extends Model

{

    protected $guarded=[];

    public function employees()

    {

        return $this->belongsToMany(\App\Employee::class)->withTimestamps();

    }

    public function revenues()

    {

        return $this->hasMany(\app\Revenue::class);

    }

}

class Employee extends Model

{

    protected $guarded=[];

    public function films()

    {

       return $this->belongsToMany(\app\Film::class)->withTimestamps();

    }

    public function roles()

    {

        return $this->belongsToMany(\App\EmployeeFilmRole::class)->withTimestamps();

    }

    public function payments()

    {

        return $this->hasMany(\App\Payment::class)->withTimestamps();

    }

    public function contracts()

    {

        return $this->hasMany(\App\Contract::class);

    }

}

class Contract extends Model

{

    protected $guarded=[];

    public function employee()

    {

        return $this->belongsTo(\App\Employee::class);

    }

    public function contract\_details()

    {

        return $this->hasMany(\App\ContractDetail::class);

    }

}

class ContractDetail extends Model

{

    protected $guarded=[];

    public function contract()

    {

        return $this->belongsTo(\App\Contract::class);

    }

    public function payments()

    {

        return $this->hasMany(\App\Payment::class);

    }

}

class Role extends Model

{

    protected $guarded=[];

    public function employees()

    {

        return $this->belongsToMany(\App\Employee::class)->withPivot(['name'])->withTimestamps();

    }

    public function role\_type()

    {

        return $this->belongsTo(\App\RoleType::class);

    }

}

class RoleTypes extends Model

{

    protected $guarded=[];

    public function roles()

    {

        return $this->hasMany(\App\Roles::class);

    }

}

**3. Write a method that, given a year, month and employee ID, will calculate the amount (if any) owed to that employee for that month.**

SELECT c.employee\_id,p.year,p.month, sum(p.amount) FROM payments p

inner join contract\_details cd on cd.id =p.contract\_detail\_id

inner join contracts c on c.id=cd.contract\_id

where c.employee\_id=2

and p.year=2020

and p.month=8

group by c.employee\_id, p.year,p.month