Model

Basic Usage   
// Defining An Eloquent Model  
class User extends Model {}  
// generate Eloquent models  
php artisan make:model User  
// specify a custom table name  
class User extends Model {  
protected $table = 'my\_users';  
}

Model::create(array('key' => 'value'));  
// Find first matching record by attributes or create  
Model::firstOrCreate(array('key' => 'value'));  
// Find first record by attributes or instantiate  
Model::firstOrNew(array('key' => 'value'));  
// Create or update a record matching attibutes, and fill with values  
Model::updateOrCreate(array('search\_key' => 'search\_value'), array('key' => 'value'));  
// Fill a model with an array of attributes, beware of mass assignment!  
Model::fill($attributes);  
Model::destroy(1);  
Model::all();  
Model::find(1);  
// Find using dual primary key  
Model::find(array('first', 'last'));  
// Throw an exception if the lookup fails  
Model::findOrFail(1);  
// Find using dual primary key and throw exception if the lookup fails  
Model::findOrFail(array('first', 'last'));  
Model::where('foo', '=', 'bar')->get();  
Model::where('foo', '=', 'bar')->first();  
Model::where('foo', '=', 'bar')->exists();  
// dynamic  
Model::whereFoo('bar')->first();  
// Throw an exception if the lookup fails  
Model::where('foo', '=', 'bar')->firstOrFail();  
Model::where('foo', '=', 'bar')->count();  
Model::where('foo', '=', 'bar')->delete();  
//Output raw query  
Model::where('foo', '=', 'bar')->toSql();  
Model::whereRaw('foo = bar and cars = 2', array(20))->get();  
Model::on('connection-name')->find(1);  
Model::with('relation')->get();  
Model::all()->take(10);  
Model::all()->skip(10);

// Default Eloquent sort is ascendant  
Model::all()->orderBy('column');  
Model::all()->orderBy('column','desc');

Soft Delete   
Model::withTrashed()->where('cars', 2)->get();  
// Include the soft deleted models in the results  
Model::withTrashed()->where('cars', 2)->restore();  
Model::where('cars', 2)->forceDelete();  
// Force the result set to only included soft deletes  
Model::onlyTrashed()->where('cars', 2)->get();

Relationships  
// One To One - User::phone()  
return $this->hasOne('App\Phone', 'foreign\_key', 'local\_key');  
// One To One - Phone::user(), The Inverse Of The Relation  
return $this->belongsTo('App\User', 'foreign\_key', 'other\_key');  
// One To Many - Post::comments()  
return $this->hasMany('App\Comment', 'foreign\_key', 'local\_key');  
// One To Many - Comment::post()  
return $this->belongsTo('App\Post', 'foreign\_key', 'other\_key');  
// Many To Many - User::roles();  
return $this->belongsToMany('App\Role', 'user\_roles', 'user\_id', 'role\_id');  
// Many To Many - Role::users();  
return $this->belongsToMany('App\User');  
// Many To Many - Retrieving Intermediate Table Columns  
$role->pivot->created\_at;  
// Many To Many - Pivot table with extra attributes  
return $this->belongsToMany('App\Role')->withPivot('column1', 'column2');  
// Many To Many - Automatically maintained created\_at and updated\_at timestamps  
return $this->belongsToMany('App\Role')->withTimestamps();  
// Has Many Through - Country::posts(), A Country model have  
// many Post models through an intermediate User model (User::country\_id)  
return $this->hasManyThrough('App\Post', 'App\User', 'country\_id', 'user\_id');  
// Polymorphic Relations - Photo::imageable()  
return $this->morphTo();  
// Polymorphic Relations - Staff::photos()  
return $this->morphMany('App\Photo', 'imageable');  
// Polymorphic Relations - Product::photos()  
return $this->morphMany('App\Photo', 'imageable');

// Polymorphic Relations - Register the morphMap in your AppServiceProvider  
Relation::morphMap([  
'Post' => App\Post::class,  
'Comment' => App\Comment::class,  
]);

// Many To Many Polymorphic Relations - Tables: posts,videos,tags,taggables  
// Post::tags()  
return $this->morphToMany('App\Tag', 'taggable');  
// Video::tags()  
return $this->morphToMany('App\Tag', 'taggable');  
// Tag::posts()  
return $this->morphedByMany('App\Post', 'taggable');  
// Tag::videos()  
return $this->morphedByMany('App\Video', 'taggable');

// Querying Relations  
$user->posts()->where('active', 1)->get();  
// Retrieve all posts that have at least one comment...  
$posts = App\Post::has('comments')->get();  
// Retrieve all posts that have three or more comments...  
$posts = Post::has('comments', '>=', 3)->get();  
// Retrieve all posts that have at least one comment with votes...  
$posts = Post::has('comments.votes')->get();  
// Retrieve all posts with at least one comment containing words like foo%  
$posts = Post::whereHas('comments', function ($query) {  
$query->where('content', 'like', 'foo%');  
})->get();

// Eager Loading  
$books = App\Book::with('author')->get();  
$books = App\Book::with('author', 'publisher')->get();  
$books = App\Book::with('author.contacts')->get();  
// Lazy Eager Loading  
$books->load('author', 'publisher');  
// Inserting Related Models  
$comment = new App\Comment(['message' => 'A new comment.']);  
$post->comments()->save($comment);  
// save multiple related models  
$post->comments()->saveMany([  
new App\Comment(['message' => 'A new comment.']),  
new App\Comment(['message' => 'Another comment.']),  
]);  
$post->comments()->create(['message' => 'A new comment.']);

// Updating a belongsTo relationship  
$user->account()->associate($account);  
$user->save();  
$user->account()->dissociate();  
$user->save();  
// Inserting Related Models - Many To Many Relationships  
$user->roles()->attach($roleId);  
$user->roles()->attach($roleId, ['expires' => $expires]);  
// Detach a single role from the user...  
$user->roles()->detach($roleId);  
// Detach all roles from the user...  
$user->roles()->detach();  
$user->roles()->detach([1, 2, 3]);  
$user->roles()->attach([1 => ['expires' => $expires], 2, 3]);  
// Any IDs that are not in the given array will be removed from the intermediate table.  
$user->roles()->sync([1, 2, 3]);  
// You may also pass additional intermediate table values with the IDs:  
$user->roles()->sync([1 => ['expires' => true], 2, 3]);

Events  
Model::creating(function($model){});  
Model::created(function($model){});  
Model::updating(function($model){});  
Model::updated(function($model){});  
Model::saving(function($model){});  
Model::saved(function($model){});  
Model::deleting(function($model){});  
Model::deleted(function($model){});  
Model::observe(new FooObserver);

Eloquent Configuration  
// Disables mass assignment exceptions from being thrown from model inserts and updates  
Eloquent::unguard();  
// Renables any ability to throw mass assignment exceptions  
Eloquent::reguard();