

Laravel

Models - Eloquent ORM - REST API howto

Eloquent

Eloquent è l' ORM integrato nel framework di Laravel. Consente di interagire con le tabelle del database in modo orientato agli oggetti, utilizzando il pattern **ActiveRecord** .

Una singola classe di modello di solito si associa ad una singola tabella di database, e possono anche essere definite relazioni di tipi diversi (uno-a-uno , uno-a-molti , molti-a-molti , polimorfico) tra diverse classi di modelli.

<https://laravel.com/docs/8.x/eloquent>

Active Record Pattern

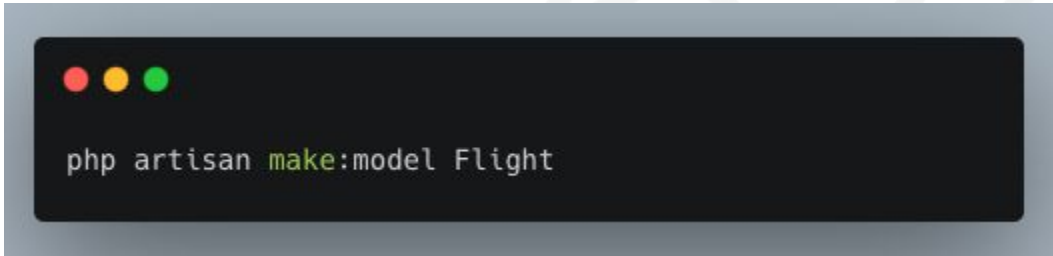
From Wikipedia, the free encyclopedia

In [software engineering](#), the **active record pattern** is an [architectural pattern](#) found in software that stores in-memory object data in [relational databases](#). It was named by [Martin Fowler](#) in his 2003 book *Patterns of Enterprise Application Architecture*.^[1] The interface of an object conforming to this pattern would include functions such as Insert, Update, and Delete, plus properties that correspond more or less directly to the columns in the underlying database table.

The active record pattern is an approach to accessing data in a [database](#). A [database table](#) or [view](#) is wrapped into a [class](#). Thus, an [object](#) instance is tied to a single row in the table. After creation of an object, a new row is added to the table upon save. Any object loaded gets its information from the database. When an object is updated, the corresponding row in the table is also updated. The wrapper class implements [accessor methods](#) or properties for each column in the table or view.

Creare un Model

Per creare un nuovo modello, esiste un apposito comando Artisan

A terminal window with a dark background and a light gray border. In the top-left corner, there are three colored circles: red, yellow, and green. The text 'php artisan make:model Flight' is displayed in a light gray monospace font.

```
php artisan make:model Flight
```

Model

- Un modello Laravel appena creato, collegato ad una semplice tabella, è già direttamente utilizzabile all'interno di un controller.
- Altrimenti, è possibile definirlo meglio con relazioni e particolari funzionalità chiamate Accessors e Mutators.
- La convenzione prevede l'uso di "snake_case" pluralizzato per i nomi delle tabelle e singolare "StudlyCase" per i nomi dei modelli. Per esempio:
 - Una tabella di cats avrebbe un modello Cat
 - Una tabella jungle_cats avrebbe un modello JungleCat
 - Una tabella users avrebbe un modello User
 - Una tabella people avrebbe un modello Person

Un modello base

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Flight extends Model
{
    protected $table = 'my_flights';
    protected $fillable = ['departure']; //Or $guarded
    protected $dates = ['seen_at'];
    //relationships here
}
```

Opzionale: si
può specificare il
nome della
tabella (se non
standard)

Selezione

```
//Singolo risultato, per ID
$user = User::find(1);
//Singolo risultato, con condizione where, di cui viene preso il primo record
$user = User::where('id', 1)->first();
//Singolo risultato per ID, se fallisce viene generato un errore 404
$user = User::findOrFail(1);
//Singolo risultato, condizione where, di cui viene preso il primo record.
//se fallisce viene generato un errore 404
$user = User::where('id', 1)->first();

//Più risultati
$users = User::get();
```

Selezione

Attraverso Eloquent è possibile comporre le proprie query, utilizzando il Query Builder.

Condizioni where, selezioni, ordinamenti, raggruppamenti: molte operazioni disponibili nel linguaggio SQL sono possibili con Eloquent, utilizzando una specifica sintassi.

Per tutto ciò che va oltre alle potenzialità di Eloquent, è possibile scrivere delle query “dirette” (raw) in linguaggio SQL.


<https://laravel.com/docs/8.x/queries>

Inserimento / aggiornamento

Oltre a leggere i dati con Eloquent, un modello può essere usato per essere creato (insert) o per aggiornarne i dati (update) con il metodo `save()` .

A seconda che venga richiamato su una nuova istanza di modello, verrà inserito il record; in caso contrario, se si è recuperato un modello dal database e si impostano nuovi valori, questo verrà aggiornato .

Inserimento - primo metodo

A terminal window with a dark background and a light gray border. It features three colored window control buttons (red, yellow, green) in the top-left corner. The terminal contains PHP code for creating a new user.

```
$user = new User();  
$user->first_name = 'John';  
$user->last_name = 'Doe';  
$user->email = 'john.doe@example.com';  
$user->password = bcrypt('my_password');  
$user->save();
```

Inserimento - secondo metodo



```
User::create([
  'first_name' => 'John',
  'last_name'  => 'Doe',
  'email'      => 'john.doe@example.com',
  'password'   => bcrypt('changeme')
]);
```

Inserimento - secondo metodo - addendum

Quando si utilizza il metodo di creazione, gli attributi interessati dall'operazione devono essere dichiarati nell'array fillable all'interno del modello:

```
class User extends Model {  
    protected $fillable = [ 'first_name', 'last_name', 'email', 'password', ];  
}
```

Aggiornamento

Per effettuare un'operazione di update, è necessario selezionare il singolo record, modificarlo ed infine salvarlo.


```
$user = User::find(1);  
$user->password = 'my_new_password';  
$user->save();
```

È inoltre possibile aggiornare uno o più modelli senza prima averli selezionati:

```
User::where('id', '>', 2)->update(['location' => 'xyz']);
```

Eliminazione - 1

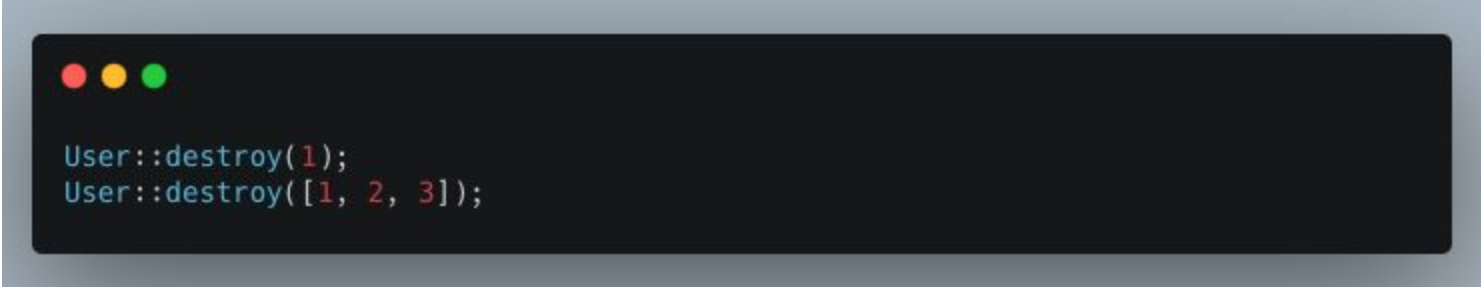
È possibile eliminare un singolo record (se selezionato come visto in precedenza) o eliminare i record risultanti da una query di selezione



```
$user = User::find(1);  
$user->delete();
```

Eliminazione - 2

In alternativa, è possibile specificare una chiave primaria (o un array di chiavi primarie) dei record che si desidera eliminare tramite il metodo `destroy()` :

A terminal window with a dark background and a light gray border. It features three colored window control buttons (red, yellow, green) in the top-left corner. The terminal contains two lines of Ruby code: `User::destroy(1);` and `User::destroy([1, 2, 3]);`. The numbers 1, 2, and 3 are highlighted in red in the original image.

```
User::destroy(1);  
User::destroy([1, 2, 3]);
```

Eliminazione - 3

Eliminare i record risultanti da una query



```
User::where('age', '<', 21)->delete();
```


Alcune features aggiuntive

- Chunking results
- Not find exception (findOrFail/firstOrFail)
- Soft delete
- Events (Observers)

<https://laravel.com/docs/8.x/eloquent>



REST API con Laravel

REST API con Laravel

A questo punto, è possibile utilizzare i concetti appresi, nello specifico:

- routes
- request / response
- controllers
- input validation
- migrations
- models (Eloquent)

per sviluppare delle REST API.



REST API con Laravel

1. Impostare correttamente le rotte corrispondenti alle azioni delle REST API (get, post, put/patch, delete)
2. Progettare migration relative alle risorse (tabelle) necessarie
3. Creare model relativi alle risorse (tabelle) necessarie
4. Implementare le azioni in un controller
 - a. Leggere e validare l'input
 - b. leggere / inserire / modificare / eliminare i record interessati dall'azione
 - c. Creare una risposta in JSON coerente con la dottrina delle REST API, anche relativamente agli HTTP status code



Q & A