Laravel Eloquent Relationship

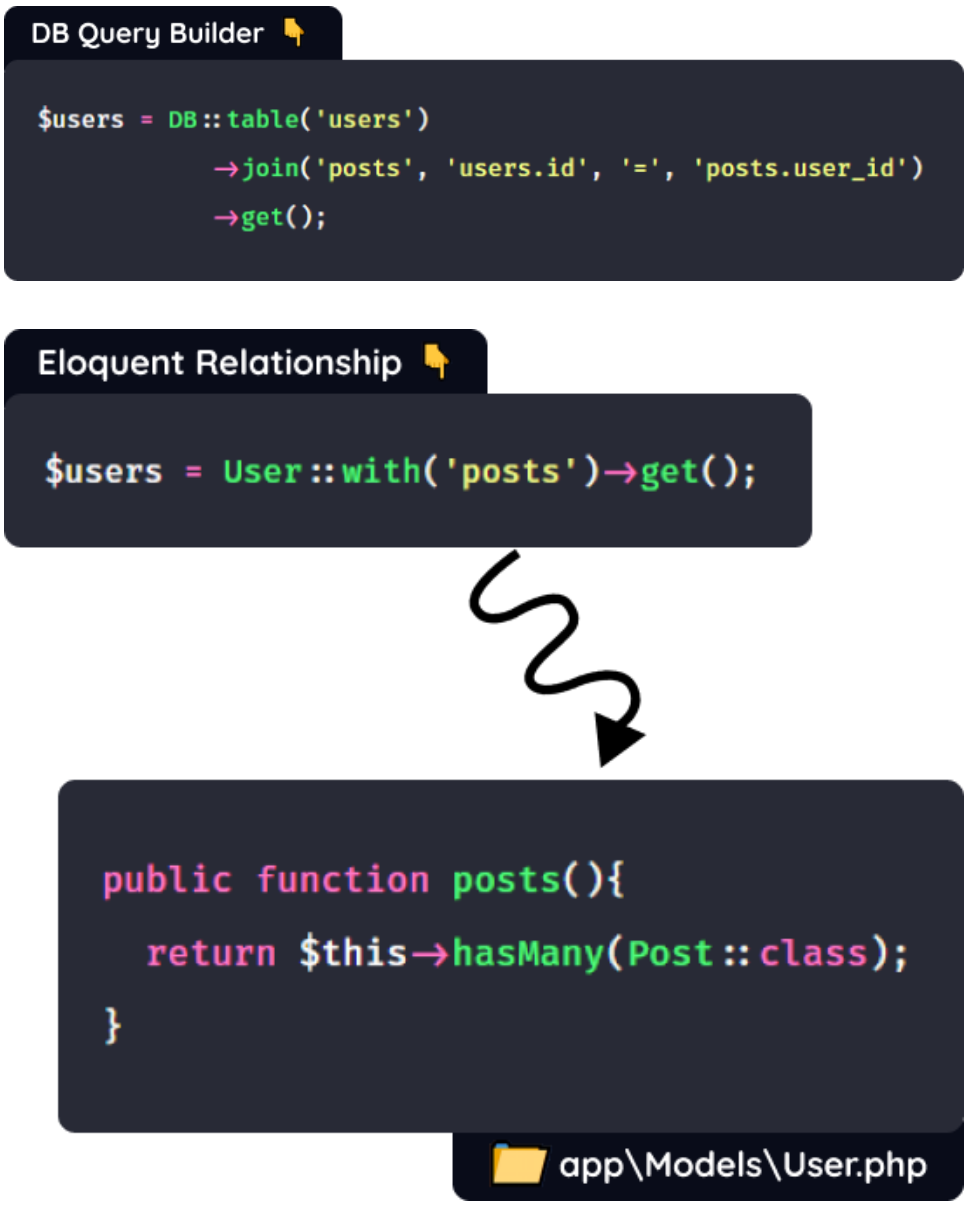
Pernah gak sih ketika kamu mau belajar sesuatu tapi sulit untuk mencari resource materi yang benar-benar *digging deeper*, dan juga enak untuk dipelajari. Terutama ketika belajar pemrograman, rata-rata materi seperti itu didapat dari orang-orang luar sana, ya mau nggak mau harus menguasai bahasa inggris.

Tetapi pada kesempatan kali ini saya mencoba untuk tampil beda dalam membuat *mini-book* tentang **Laravel Eloquent Relationship** supaya mudah dicerna walaupun penjelasannya detail, karena *mini-book* ini memuat visualisasi yang memudahkan kamu dalam memahami materi. Penasaran? Langsung aja dibaca 😀.

Table of Content

Introduction

Tahu gak kamu kalau di laravel mempunyai dua cara dalam merelasikan tabel yang telah dibuat. Yang pertama dengan cara *Eloquent* (menggunakan model) dan yang kedua dengan cara merelasikannya lewat *Database Query Builder*. Hmmmm..... apa sih perbedaan keduanya? bisa kalian cek di bawah 👇

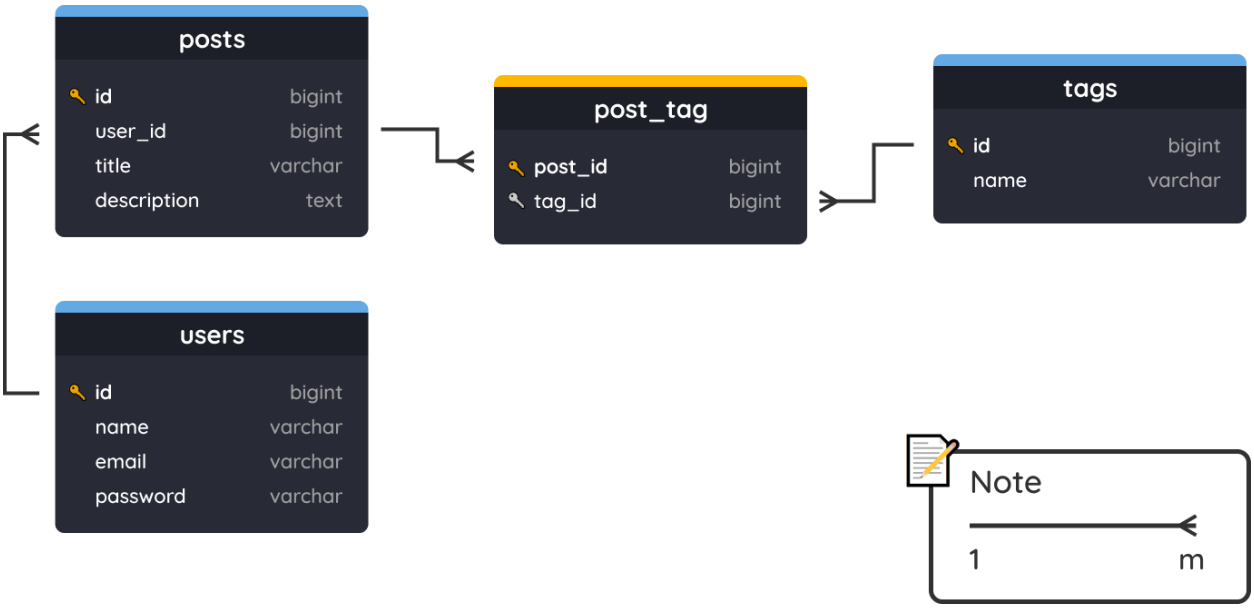


Tampak lebih pendek bukan jika kita menggunakan *"eloquent"* sebagai relasi dari tabel-tabel yang kita punya di dalam database 😀. Ya, karena *eloquent* menggunakan model yang bersangkutan untuk merelasikan tabel yang kita punya.

Sebelum Berpetualang

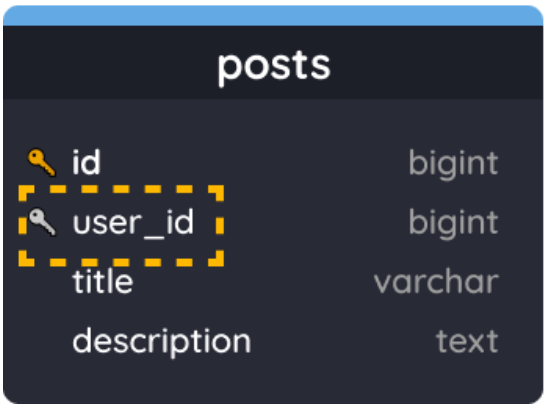
Sebelum bertualang untuk memulai perjalanan yang akan kita lalui, saya bakalan memberikan gambaran tabel apa saja yang akan kita miliki kedepannya. Tentunya

seiring berjalannya perjalanan kita, nanti kamu pasti bakal ketemu dengan tabel-tabel berikut ini.

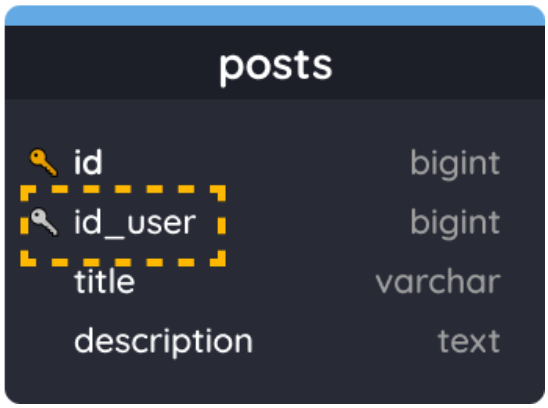


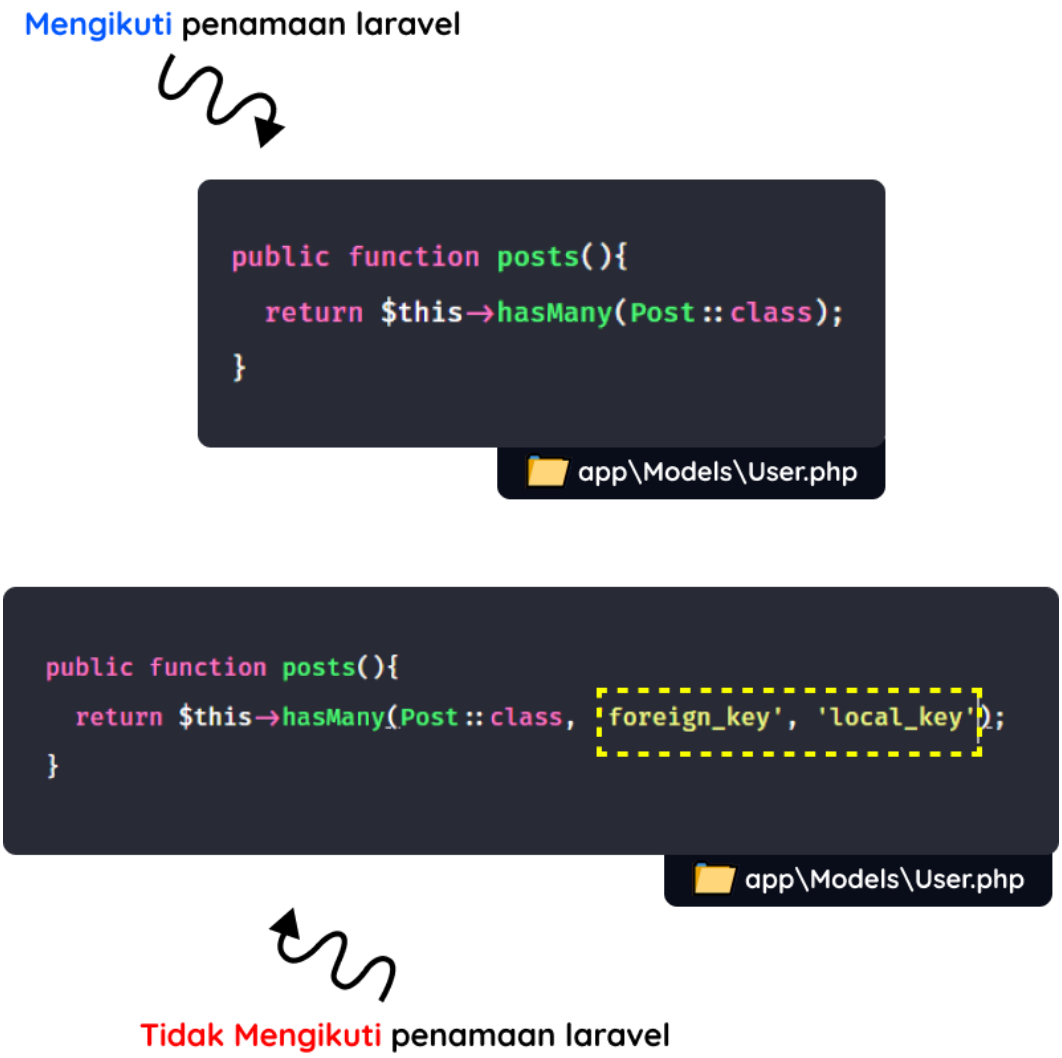
Owh iya, jika kamu nantinya tidak mengikuti "**penamaan**" laravel untuk membuat nama *field* di dalam tabelnya, maka kamu harus menambahkan beberapa *parameter* di dalam *functionnya* nanti, sebagai contoh seperti ini :

Mengikuti penamaan laravel



Tidak Mengikuti penamaan laravel



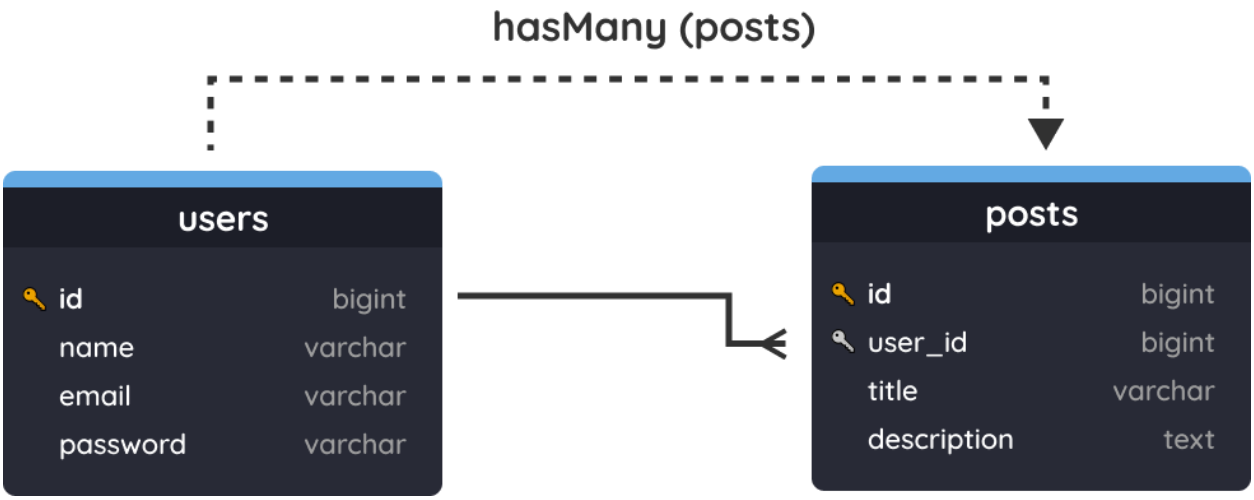


Tapi tenang aja, kalau kamu nggak ngikutin penamaan laravel, nantinya kita bakalan *digging deeper* juga mengenai "**key**" dan seperti apa gambarannya di dalam tabel, seperti *foreign_key*, *local_key* dan sebagainya. Oke untuk itu langsung saja *let's start our journey* 🤩.

A. One to Many

Sebelum mempelajari konsep *One to Many*, disini kita siapin terlebih dahulu dua tabel yang bakalan kita pakai yaitu tabel **users** dan tabel **posts**, karena satu user

bisa memiliki banyak postingan (*One to Many*).

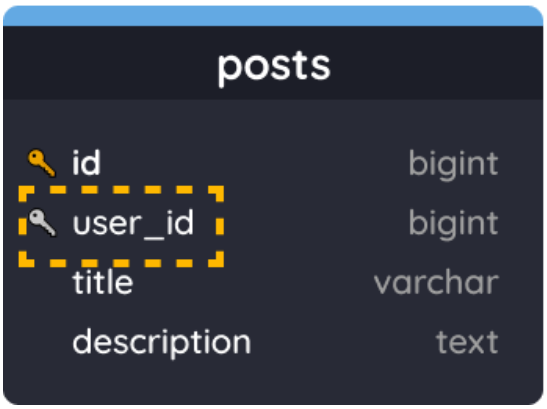


Bisa kamu lihat pada gambar diatas bahwa satu *user* bisa memiliki banyak *post* (*hasMany*). Oleh karena itu di dalam **Model User** kita bisa menuliskan *function* seperti ini.

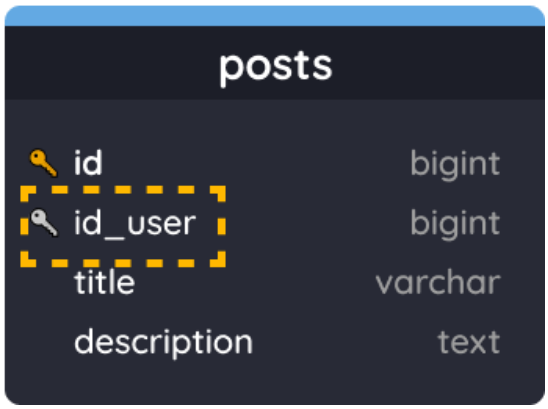
Cara diatas adalah cara ketika kamu menggunakan "penamaan" *field* di laravel, yaitu untuk *foreign_key* nya sudah kamu buat **user_id**. Tetapi bagaimana jika kamu

tidak mengikuti penamaan laravel? sebagai contoh kamu membuat *field* nya dengan nama **id_user.**

Mengikuti penamaan laravel



Tidak Mengikuti penamaan laravel

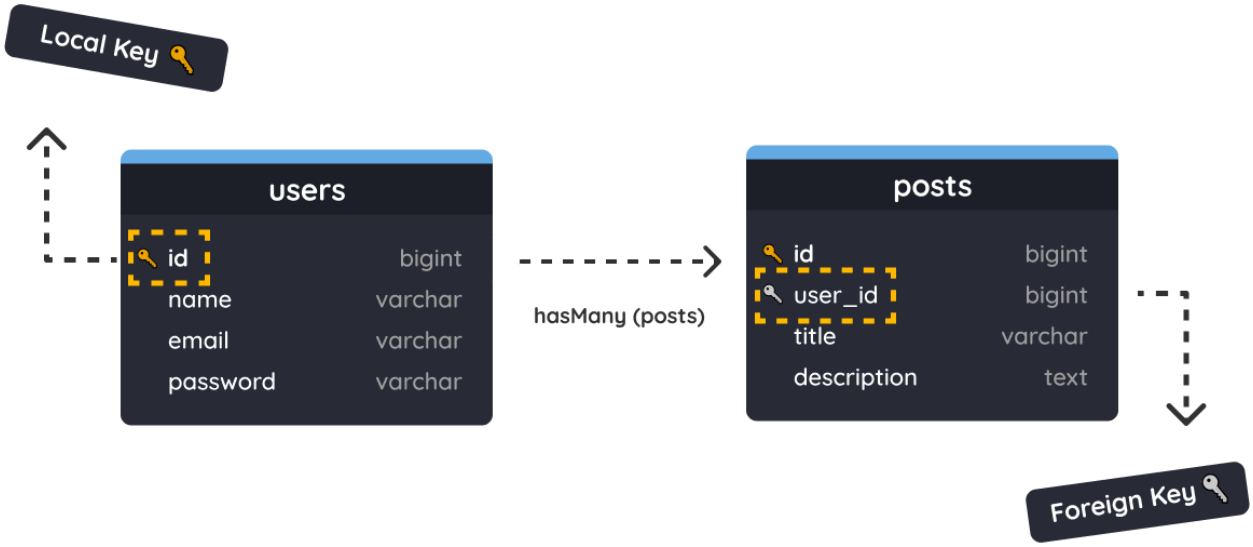


Untuk itu kamu perlu menambahkan beberapa parameter lagi di dalam *function* nya, jadinya seperti ini.

```php
public function posts(){
  return $this->hasMany(Post::class, 'foreign_key', 'local_key');
}
```

app\Models\User.php

Mungkin kamu baru pertama kali mendengar istilah *foreign_key* ataupun *local_key*, untuk mengerti hal tersebut kalian bisa melihat gambaran dibawah ini.



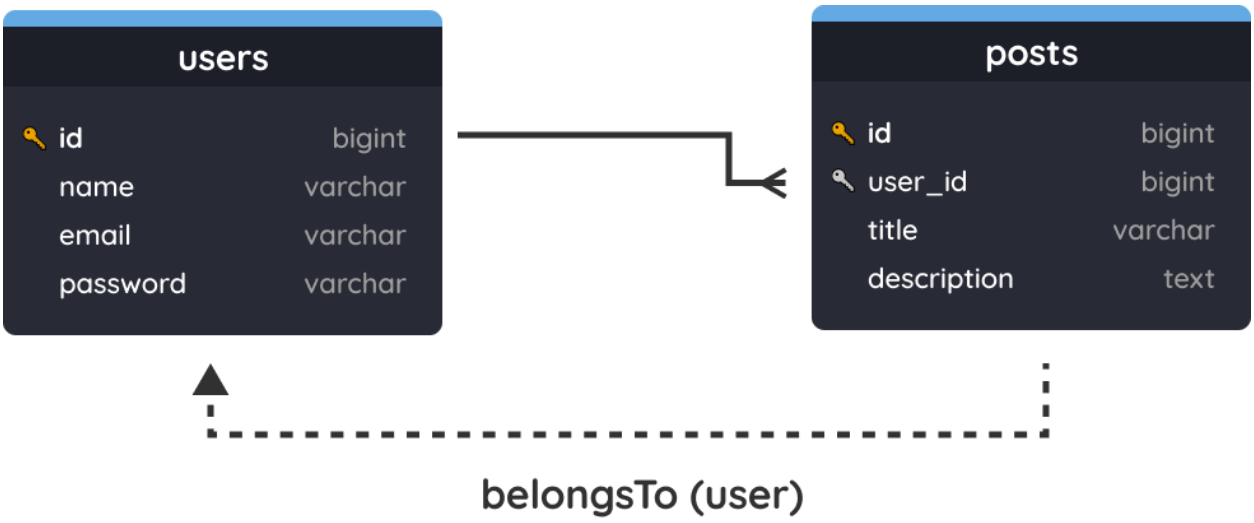
Yuppp... *foreign_key* adalah "kunci silver", dan *local_key* adalah "kunci oren". Singkatnya gitu 😂, jadi *function* yang telah dibuat sebelumnya menjadi seperti ini.

```php
public function posts(){
  return $this->hasMany(Post::class, 'user_id', 'id');
}
```

app\Models\User.php

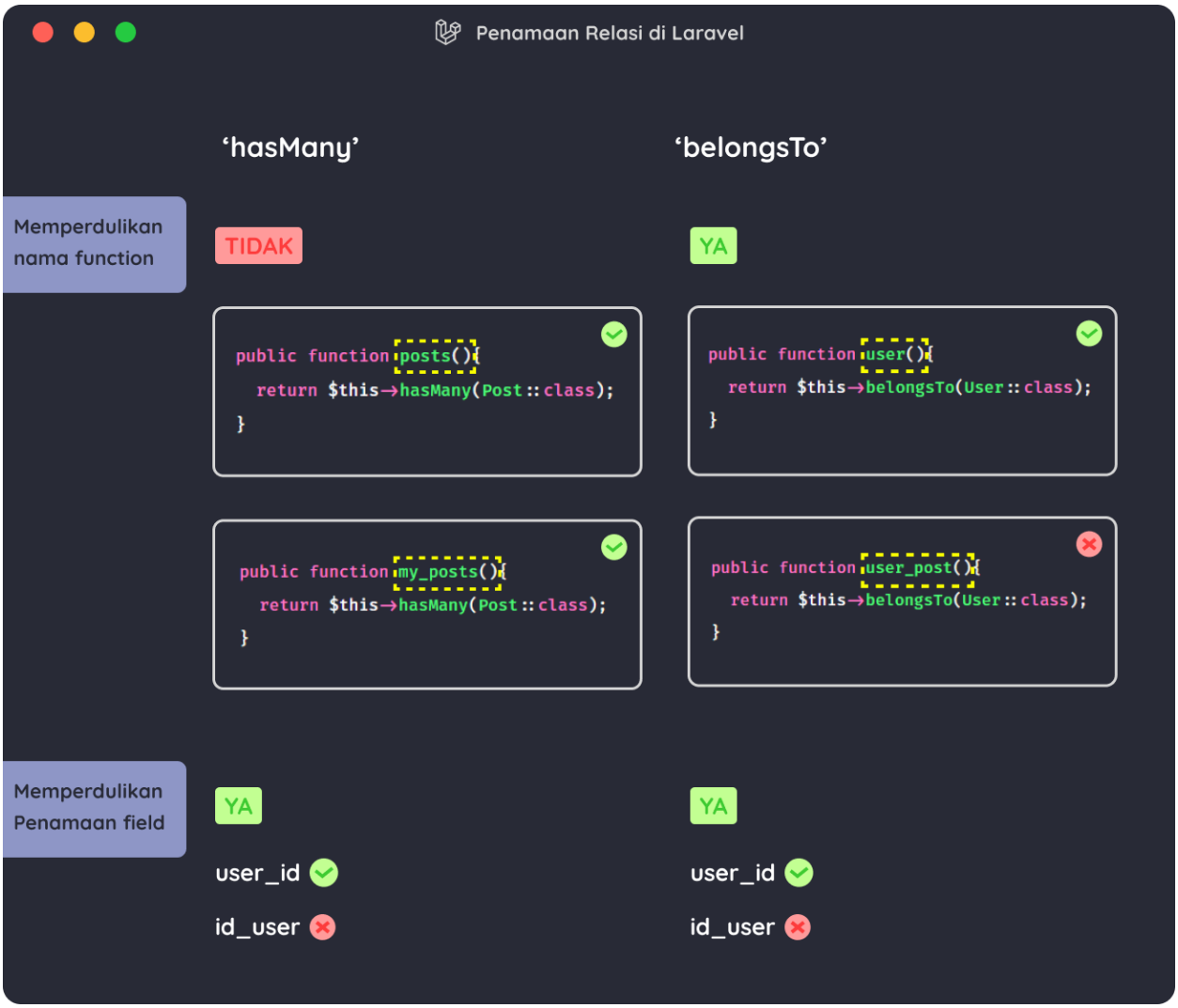
B. Many to One

Konsep *One to Many* dan *Many to One* sebenarnya sama saja cuma **dibalik**, karena konsepnya sama kita masih menggunakan dua tabel sebelumnya, yaitu tabel **users** dan tabel **posts**.



Kalau sebelumnya dari tabel **users** ke tabel **posts**, sekarang kebalikannya yaitu dari tabel *posts* ke tabel *users*, dengan relasi *post* adalah milik *user* (*belongsTo*). Oleh karena itu di dalam **Model Post** kita bisa menuliskan *function* seperti ini.

Cara diatas adalah cara ketika kamu menggunakan "penamaan" sesuai dengan laravel. Eittssss... Tahukah kamu kalau penamaan untuk **belongsTo** itu bukan hanya dilihat dari *field* dalam *database* nya saja, namun juga melihat dari penamaan *function* nya, coba deh lihat gambar dibawah.

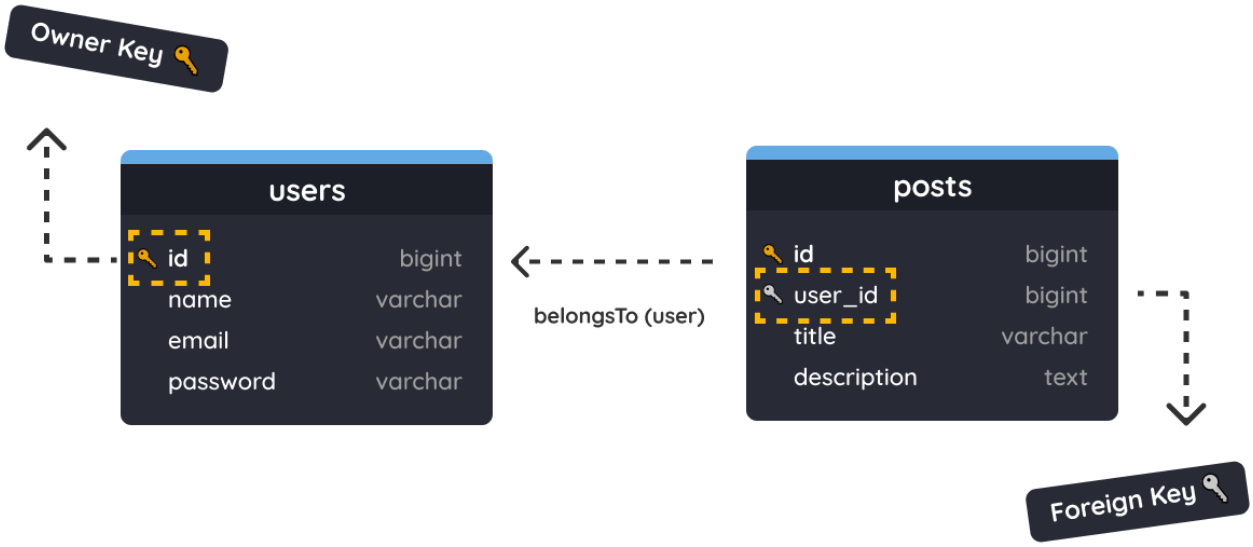


Tapi kalau kamu tidak mengikuti penamaan di laravel untuk relasi **belongsTo**, kamu bisa buat alternatif seperti ini.

```php
public function user_post(){
  return $this→belongsTo(User::class, 'foreign_key', 'owner_key');
}
```

app\Models\Post.php

Kalau sebelumnya di relasi *One to Many* ada *key* yang namanya *foreign_key*, dan *local_key*, kalau di relasi *Many to One* ada yang namanya *foreign_key*, dan *owner_key*. Sebenarnya konsepnya sama saja kok, kalian bisa lihat gambar dibawah



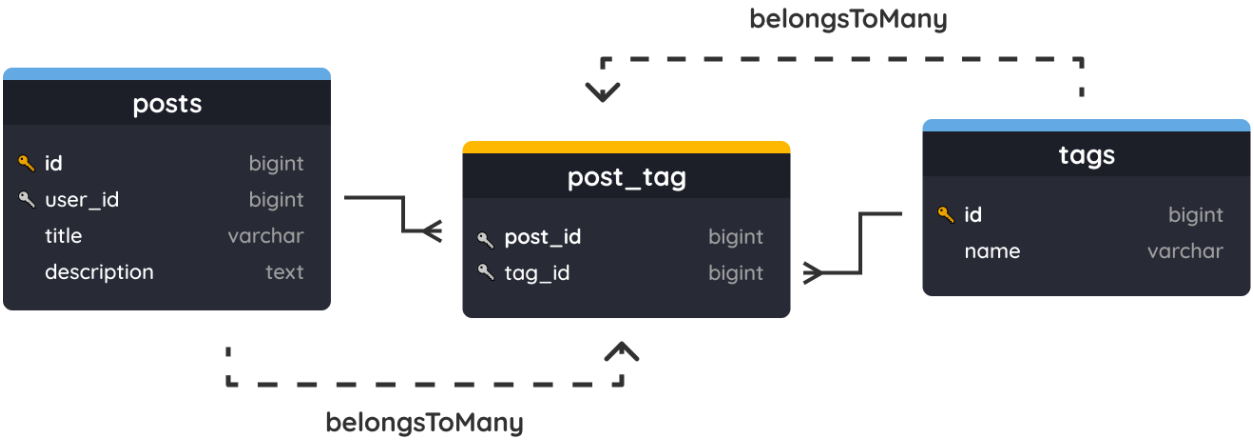
Sehingga code kamu akan menjadi seperti ini.

```php
public function user_post(){
  return $this->belongsTo(User::class, 'user_id', 'id');
}
```

app\Models\Post.php

C. Many to Many

Sekarang kita masuk ke konsep *Many to Many*, untuk tabel yang kita gunakan adalah tabel **posts**, **post_tag,** dan **tags**.



Untuk relasi *Many to Many* kira-kira sebagai berikut : satu postingan bisa memilki banyak *tag*, dan satu tag bisa memiliki banyak postingan, maka kamu harus membuat tabel **pivot**, yaitu tabel yang menjembatani keduanya.

Untuk penamaannya sendiri gunakan berdasarkan urutan alphabet, semisal *posts* (huruf pertama **'p'**), dan *tags* (huruf pertama **'t'**), maka penamaan tabelnya adalah *post_tag*, bukan *tag_post*. Untuk di dalam modelnya kamu bisa tuliskan seperti ini jika sudah mengikuti penamaan laravel.

```php
public function tags(){
    return $this->belongsToMany(Tag::class);
}
```

app\Models\Post.php

```php
public function posts(){
    return $this->belongsToMany(Post::class);
}
```

app\Models\Tag.php

Cara diatas berlaku jika kamu sudah mengikuti penamaan di laravel untuk field saja, untuk nama *function* boleh bebas dibuat apapun, kalian bisa lihat ringkasannya di bawah.


Penamaan Relasi di Laravel
‘belongsToMany’
Memperdulikan nama function
TIDAK
public function posts(){
return $this→belongsToMany(Post::class);
}
public function all_posts(){
return $this→belongsToMany(Post::class);
}
Memperdulikan Penamaan field
YA
post_id
id_post

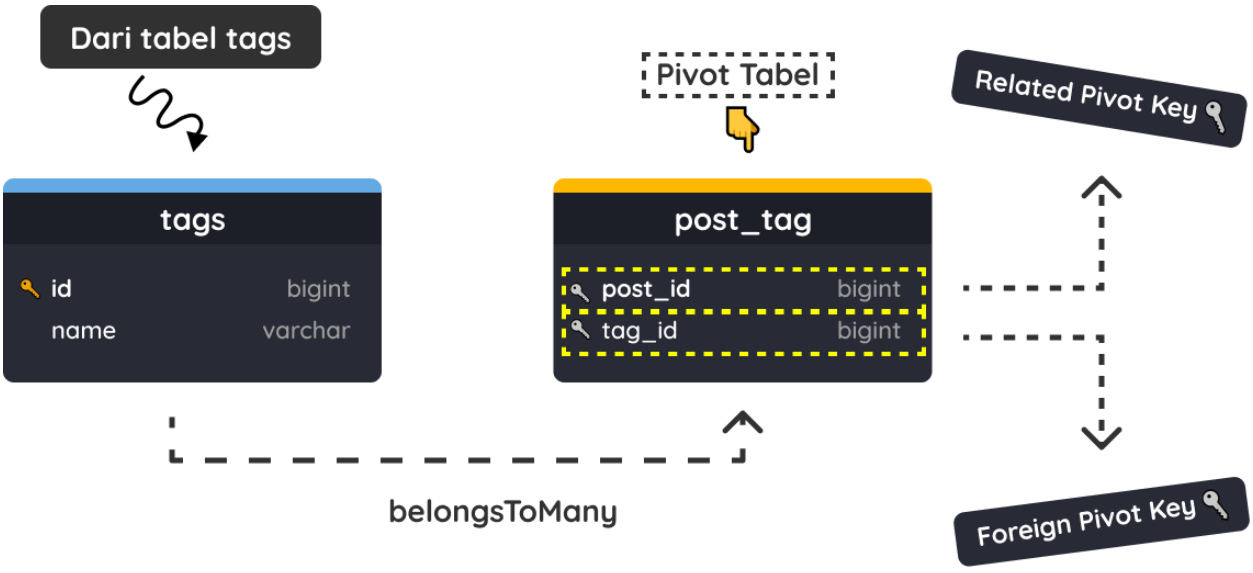
> Mungkin agak *tricky* untuk mengetahui penamaan yang benar pada laravel, tenang aja diakhir nanti bakalan saya kasih *tips* untuk kamu 🤩.

Kalau kamu tidak mengikuti penamaan laravel untuk relasi *Many to Many*, maka kamu harus menambah beberapa *parameter* seperti dibawah.

```php
public function posts(){
  return $this->belongsToMany(Post::class, 'pivot_table', 'foreign_pivot_key', 'related_pivot_key');
}
```

app\Models\Tag.php

Mungkin kamu bingung apa yang harus diisi untuk ketiga *parameter* diatas karena mungkin juga kamu baru pertama kali dengar istilah itu, penjelasan simpelnya di gambar bawah.



> Gambar diatas merupakan relasi **belongsToMany** dari tabel *tags* ke tabel *posts* ya.

Bisa kalian lihat kalau *foreign_pivot_key* itu merupakan *key* dari tabel sumber, karena kita sumbernya dari tabel *tags* maka **foreign_pivot_key** nya adalah *tag_id*. Untuk **related_pivot_key** pasti kalian sudah tahu gimana konsepnya 😀.

Dan juga kalau relasi belongsToMany dari tabel *posts* ke tabel *tags* saya rasa kamu sudah mengerti dengan konsep mana yang *foreign_pivot_key* dan mana yang *related_pivot_key*.

Sehingga code di dalam *function* menjadi seperti ini.

```php
public function posts(){
  return $this->belongsToMany(Post::class, 'post_tag', 'tag_id', 'post_id');
}
```

app\Models\Tag.php

Conclusion

Nggak kerasa petualangan kita sudah berakhir sampai disini, inti dari petualangan kali ini adalah kita belajar bagaimana menggunakan relasi dengan *Eloquent Relationship Model* (ORM), dan yang kerennya ketika menggunakan ORM kita tidak perlu menuliskan code setiap saat kita membutuhkan relasi, karena *code* tersebut kita letakkan di dalam Model, dan tentunya ini lebih *clean* daripada menggunakan *Database Query Builder*. Owh iya sebelum menutup petualangan kali ini saya bakalan memberikan bonus berupa *tips* dibawah ini.

Tips Laravel Eloquent Relationship

1. Gunakanlah penamaan laravel, yaitu perhatikan mana yang *singular* dan mana yang *plural*. Penamaan itu akan membantu kamu, karena tidak perlu menambahkan *parameter* ketika membuat relasi.
2. Jika kamu tidak terbiasa dengan penamaan laravel, dan lebih memilih penamaan sendiri maka tidak perlu menghafal parameter apa saja yang perlu

diisi ketika membuat *function*, cukup **klik disini** untuk melihat pola penamaan *key* dalam *eloquent*.

3. **PRACTICE, PRACTICE, AND PRACTICE!!**