|  |  |  |
| --- | --- | --- |
| NAMA | **:** | KARTIKA TRI JULIANA |
| PROGRAM STUDI | **:** | D4 – Sistem Informasi Bisnis |
| KELAS | **:** | 2B |
| NIM | **:** | 2341760116 |

# JOBSHEET 03

**MIGRATION, SEEDER, DB FAÇADE, QUERY BUILDER, dan ELOQUENT ORM**

Sebelumnya kita sudah membahas mengenai *Routing, Controller*, dan *View* yang ada di Laravel. Sebelum kita masuk pada pembuatan aplikasi berbasis website, alangkah baiknya kita perlu menyiapkan Basis data sebagai tempat menyimpan data-data pada aplikasi kita nanti. Selain itu, umumnya kita perlu menyiapkan juga data awal yang kita gunakan sebelum membuat aplikasi, seperti data user administrator, data pengaturan sistem, dll.

Untuk itu, kita memerlukan teknik untuk merancang/membuat table basis data sebelum membuat aplikasi. Laravel memiliki fitur dalam pengelolaan basis data seperti, migration, seeder, model, dll.

Sebelum kita masuk materi, kita buat dulu project baru yang akan kita gunakan untuk membangun aplikasi sederhana dengan topik *Point of Sales (PoS),* sesuai dengan **Studi Kasus PWL.pdf**.

Jadi kita bikin project Laravel 10 dengan nama **PWL\_POS.**

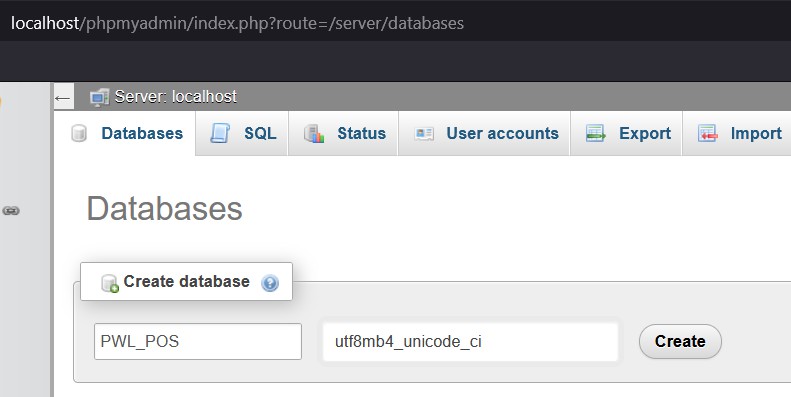
*Project* **PWL\_POS** akan kita gunakan sampai pertemuan 12 nanti, sebagai project yang akan kita pelajarai

# PENGATURAN DATABASE

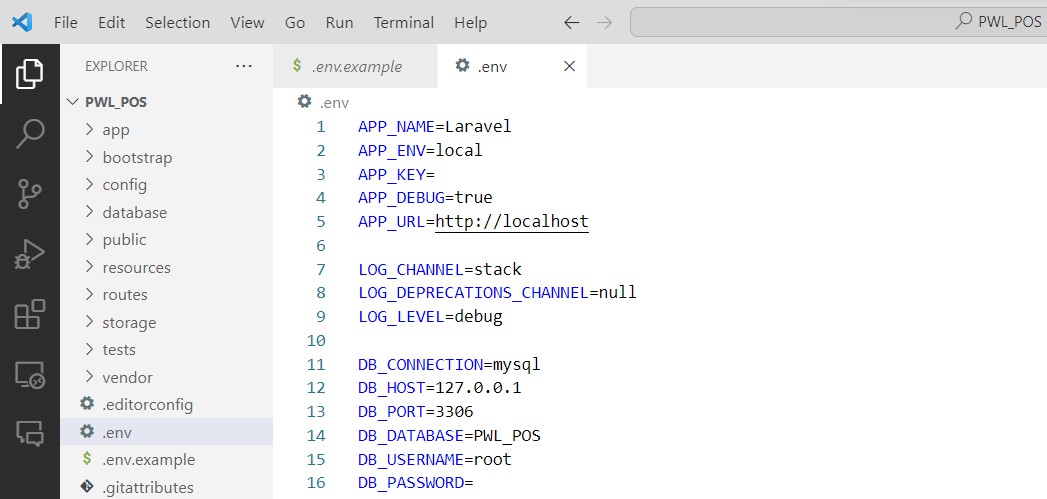
Database atau basis data menjadi komponen penting dalam membangun sistem. Hal ini dikarenakan database menjadi tempat untuk menyimpan data-data transaksi yang ada pada sistem. Koneksi ke database perlu kita atur agar sesuai dengan database yang kita gunakan.

**Praktikum 1** - pengaturan database:

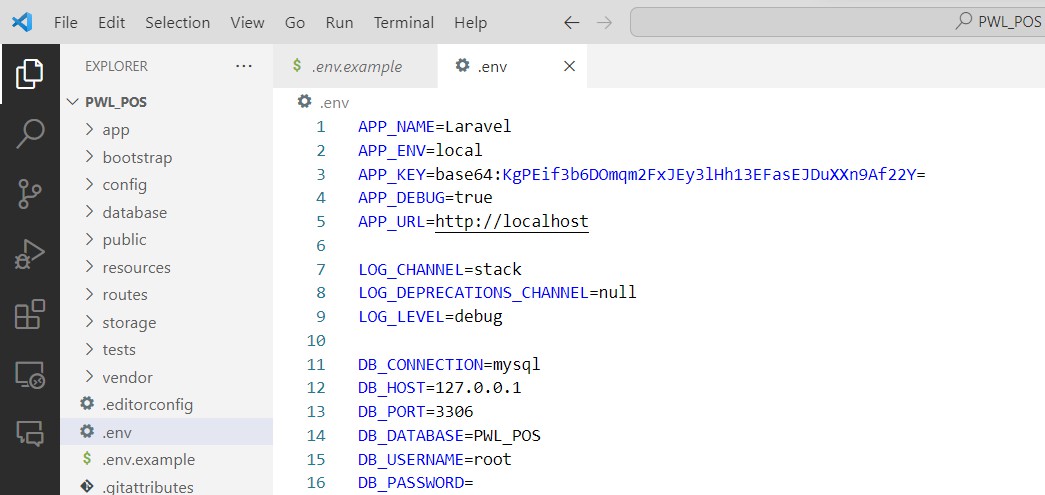
* 1. Buka aplikasi phpMyAdmin, dan buat database baru dengan nama PWL\_POS



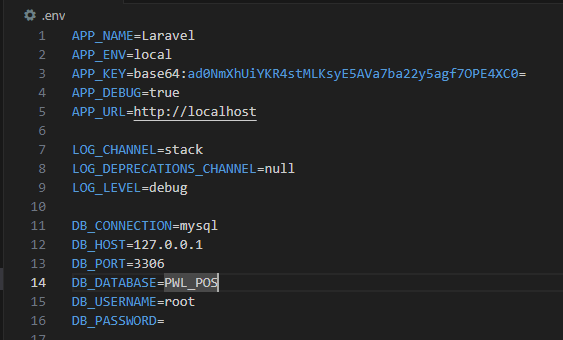
* 1. Buka aplikasi VSCode dan buka folder project PWL\_POS yang sudah kita buat
  2. Copy file .env.example menjadi .env
  3. Buka file .env, dan pastikan konfigurasi **APP\_KEY** bernilai. Jika belum bernilai silahkan kalian *generate* menggunakan php artisan.



* 1. Edit file .env dan sesuaikan dengan database yang telah dibuat



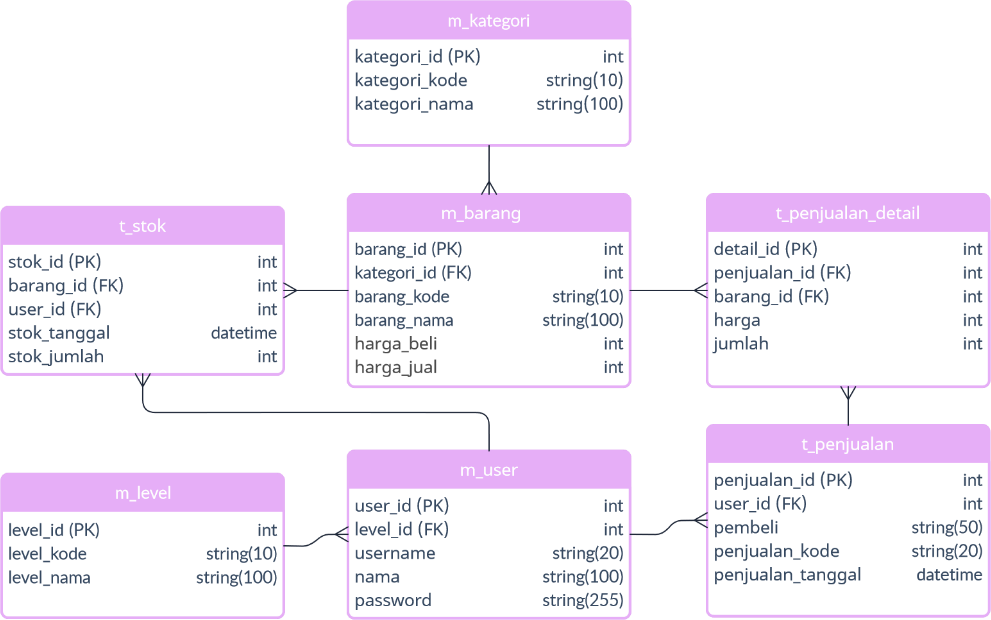
* 1. Laporkan hasil Praktikum-1 ini dan *commit* perubahan pada *git*.



# MIGRATION

Migration pada Laravel merupakan sebuah fitur yang dapat membantu kita mengelola database secara efisien dengan menggunakan kode program. Migration membantu kita dalam membuat (*create*), mengubah (*edit*), dan menghapus (*delete*) struktur tabel dan kolom pada database yang sudah kita buat dengan cepat dan mudah. Dengan Migration, kita juga dapat melakukan perubahan pada struktur database tanpa harus menghapus data yang ada.

Salah satu keunggulan menggunakan migration adalah mempermudah proses instalasi aplikasi kita, Ketika aplikasi yang kita buat akan diimplementasikan di server/komputer lain.



Sesuai dengan topik pembelajaran kita untuk membangun sistem *Point of Sales (PoS)* sederhana, maka kita perlu membuat migration sesuai desain database yang sudah didefinisikan pada file Studi Kasus PWL.pdf

Dalam membuat file migration di Laravel, yang perlu kita perhatikan adalah struktur table yang ingin kita buat.

**TIPS *MIGRATION***

Buatlah file migration untuk table yang tidak memiliki relasi (table yang tidak ada *foreign key*) dulu, dan dilanjutkan dengan membuat file migrasi yang memiliki relasi yang sedikit, dan dilanjut ke file migrasi dengan table yang memiliki relasi yang banyak.

Dari tips di atas, kita dapat melakukan cek untuk desain database yang sudah ada dengan mengetahui jumlah *foreign key* yang ada. Dan kita bisa menentukan table mana yang akan kita buat migrasinya terlebih dahulu.

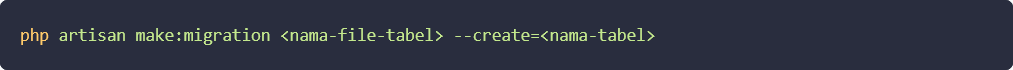
|  |  |  |
| --- | --- | --- |
| No Urut | Nama Tabel | Jumlah FK |
| 1 | m\_level | 0 |
| 2 | m\_kategori | 0 |
| 3 | m\_user | 1 |
| 4 | m\_barang | 1 |
| 5 | t\_penjualan | 1 |
| 6 | t\_stok | 2 |
| 7 | t\_penjualan\_detail | 2 |

**INFO**

Secara default Laravel sudah ada table **users** untuk menyimpan data pengguna, tapi pada praktikum ini, kita gunakan table sesuai dari file Studi Kasus PWL.pdf yaitu **m\_user.**

Pembuatan file migrasi bisa menggunakan 2 cara, yaitu

1. Menggunakan artisan untuk membuat *file migration*

**

1. Menggunakan artisan untuk membuat *file model + file migration*

**

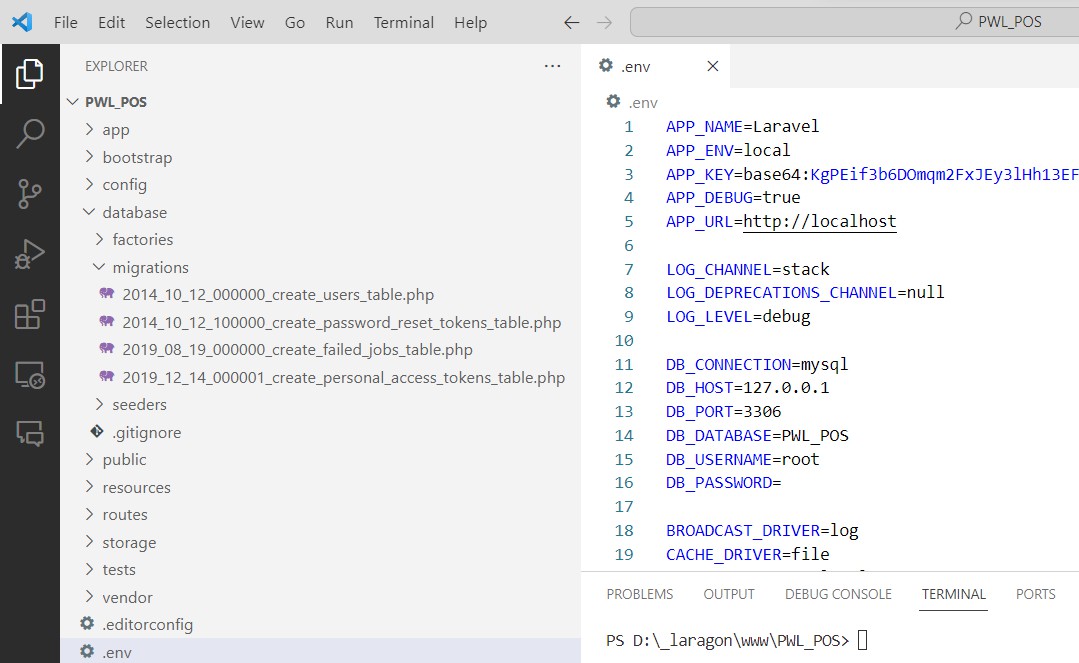
Perintah **-m** di atas adalah *shorthand* untuk opsi membuat file migrasi berdasarkan model yang dibuat.

Pada Laravel, file-file *migration* ataupun *seeder* berada pada folder **PWL\_POS/database**

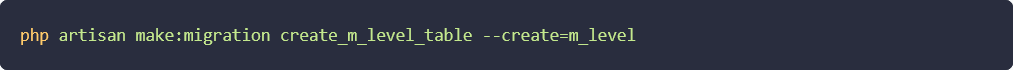


**Praktikum 2.1** - Pembuatan file migrasi tanpa relasi

1. Buka *terminal* VSCode kalian, untuk yang di kotak merah adalah default dari laravel

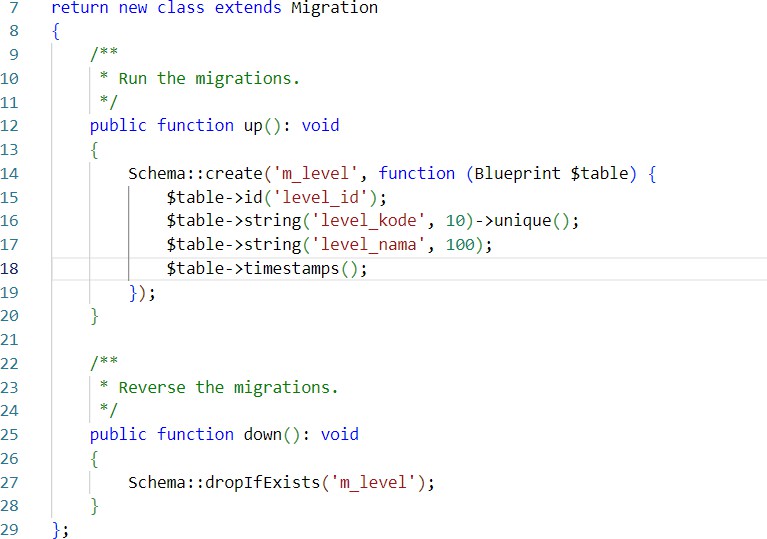


1. Kita abaikan dulu yang di kotak merah (jangan di hapus)
2. Kita buat file migrasi untuk table m\_level dengan perintah





1. Kita perhatikan bagian yang di kotak merah, bagian tersebut yang akan kita modifikasi sesuai desain database yang sudah ada



**INFO**

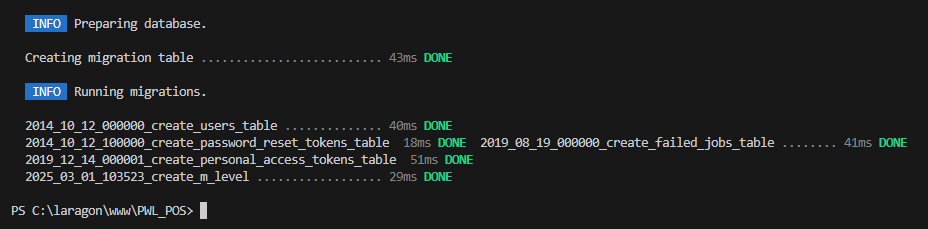
Dalam fitur migration Laravel, terdapat berbagai macam function untuk membuat kolom di table database. Silahkan cek disini

[https://laravel.com/docs/10.x/migrations#available-column-types](https://laravel.com/docs/10.x/migrations%23available-column-types)

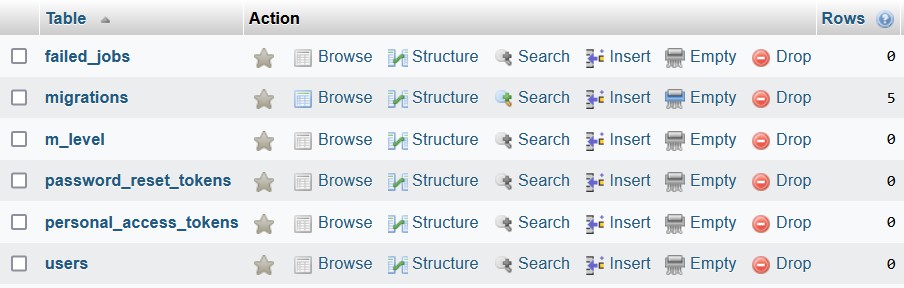
1. Simpan kode pada tahapan 4 tersebut, kemudian jalankan perintah ini pada terminal VSCode untuk melakukan migrasi







1. Kemudian kita cek di phpMyAdmin apakah table sudah ter-generate atau belum



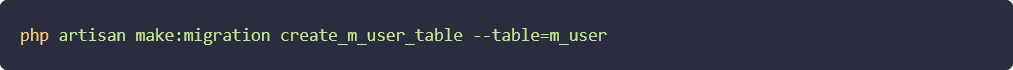
1. Ok, table sudah dibuat di database
2. Buat table *database* dengan *migration* untuk table **m\_kategori** yang sama-sama tidak memiliki *foreign key*

**

1. Laporkan hasil Praktikum-2.1 ini dan *commit* perubahan pada *git*.

**Praktikum 2.2** - Pembuatan file migrasi dengan relasi

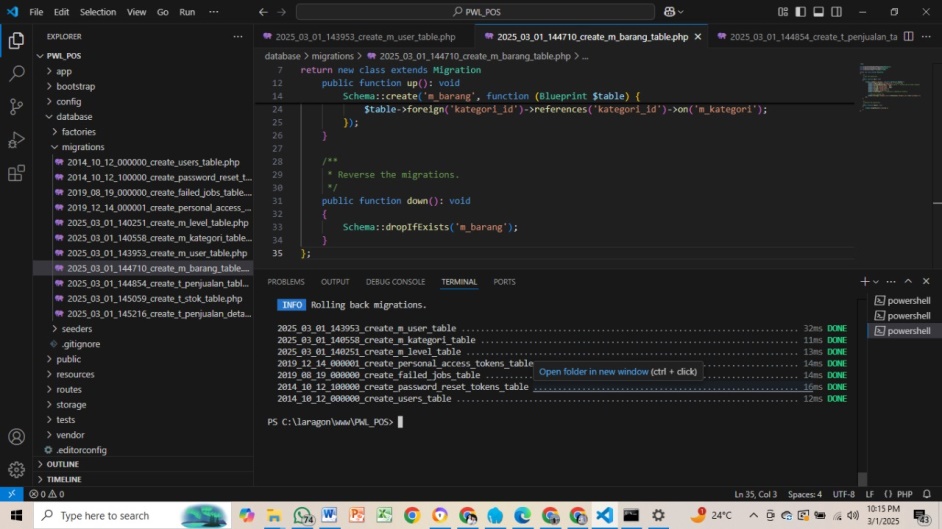
1. Buka *terminal* VSCode kalian, dan buat file migrasi untuk table **m\_user**

****

1. Buka file migrasi untuk table **m\_user**, dan modifikasi seperti berikut



1. Simpan kode program Langkah 2, dan jalankan perintah **php artisan migrate**. Amati apa yang terjadi pada database.

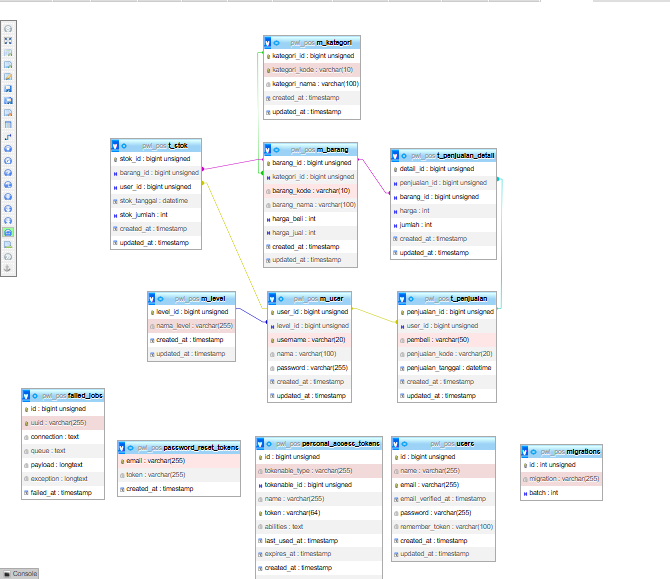


1. Buat table *database* dengan *migration* untuk table-tabel yang memiliki *foreign key*

|  |
| --- |
| m\_barang |
| t\_penjualan |
| t\_stok |
| t\_penjualan\_detail |

1. Jika semua file migrasi sudah di buat dan dijalankan maka bisa kita lihat tampilan

*designer* pada **phpMyAdmin** seperti berikut

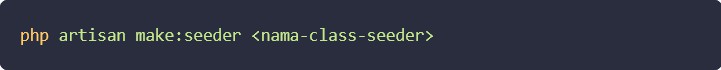


1. Laporkan hasil Praktikum-2.2 ini dan *commit* perubahan pada *git*.

# SEEDER

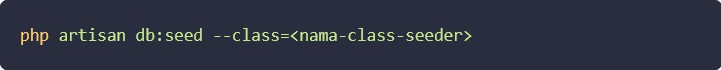
Seeder merupakan sebuah fitur yang memungkinkan kita untuk mengisi database kita dengan data awal atau data *dummy* yang telah ditentukan. Seeder memungkinkan kita untuk membuat data awal yang sama untuk setiap penggunaan dalam pembangunan aplikasi. Umumnya, data yang sering dibuat *seeder* adalah data penggunna karena data tersebut akan digunakan saat aplikasi pertama kali di jalankan dan membutuhkan aksi *login*.

* 1. Perintah umum dalam **membuat *file seeder*** adalah seperti berikut



Perintah tersebut akan men-generate file seeder pada folder **PWL\_POS/database/seeders**

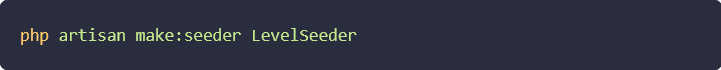
* 1. Dan perintah untuk **menjalankan *file seeder*** seperti berikut

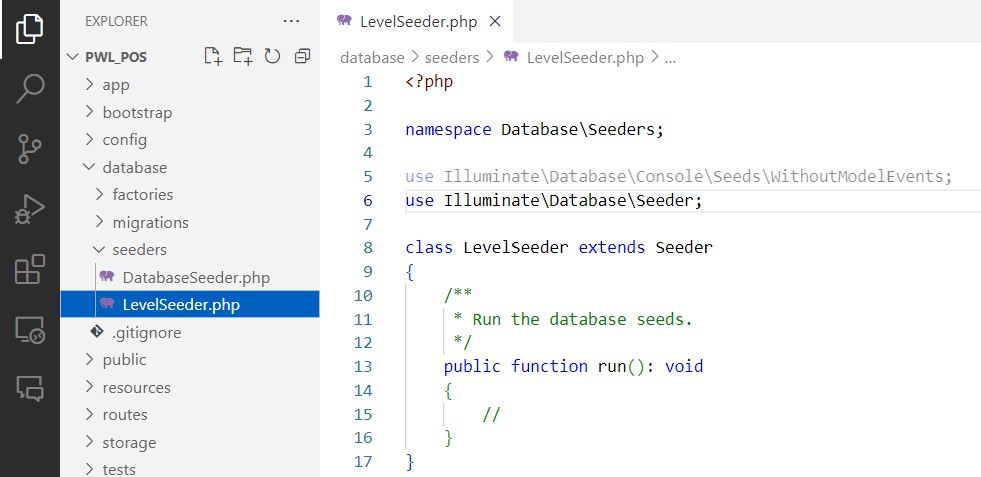


Dalam proses pengembangan suatu aplikasi, seringkali kita membutuhkan data awal tiruan atau *dummy* data untuk memudahkan pengujian dan pengembangan aplikasi kita. Sehingga fitur *seeder* bisa kita pakai dalam membuat sebuah aplikasi web.

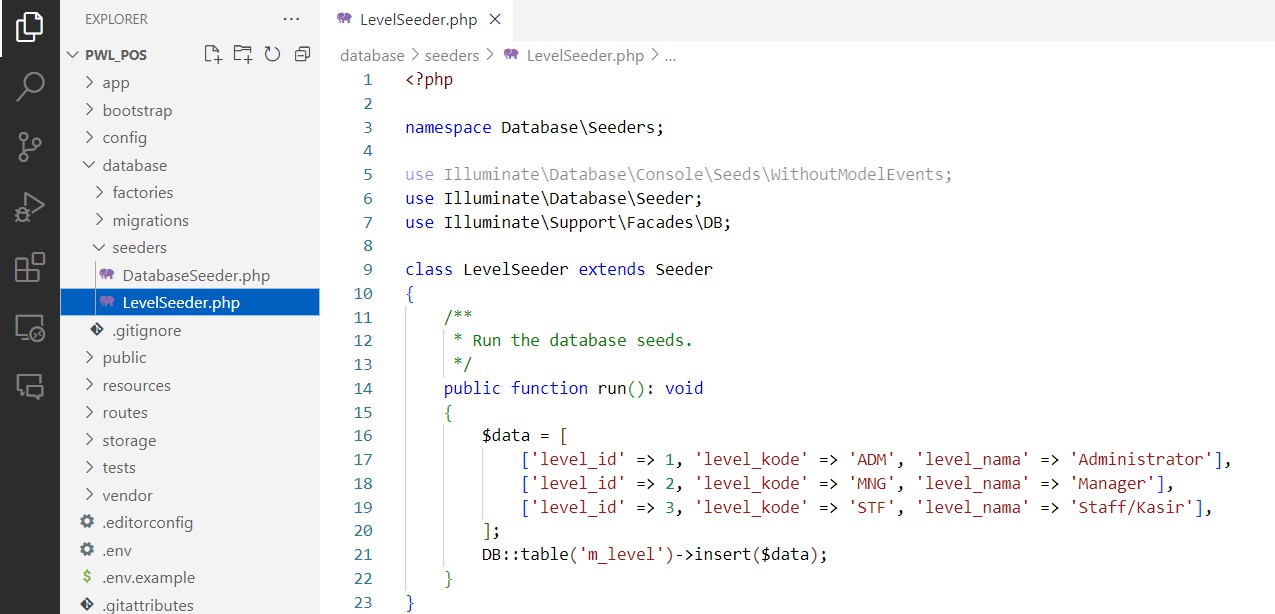
**Praktikum 3** – Membuat file *seeder*

1. Kita akan membuat file seeder untuk table m\_level dengan mengetikkan perintah



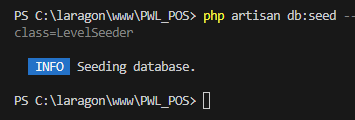


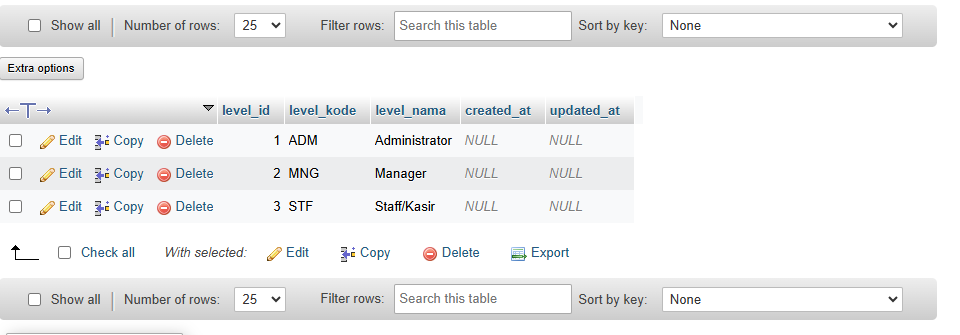
1. Selanjutnya, untuk memasukkan data awal, kita modifikasi file tersebut di dalam function run()

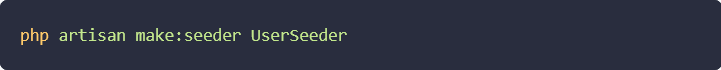


1. Selanjutnya, kita jalankan file *seeder* untuk table m\_level pada terminal





1. Ketika *seeder* berhasil dijalankan maka akan tampil data pada table m\_level
2. Sekarang kita buat file *seeder* untuk table m\_user yang me-*refer* ke table m\_level



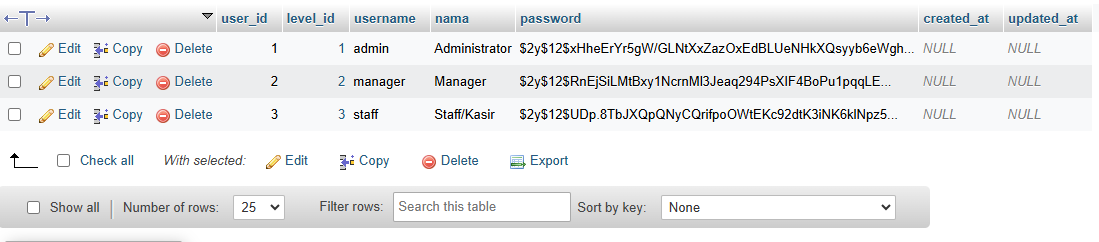
1. Modifikasi file class **UserSeeder** seperti berikut



1. Jalankan perintah untuk mengeksekusi class UserSeeder



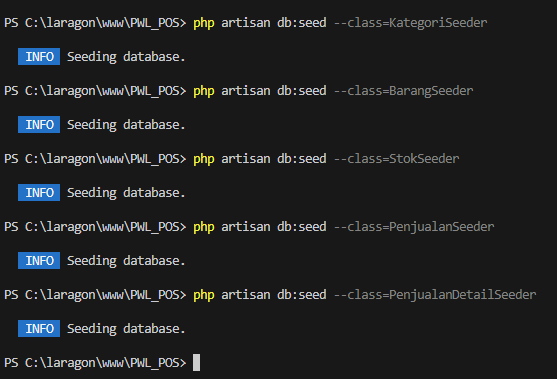
1. Perhatikan hasil seeder pada table m\_user



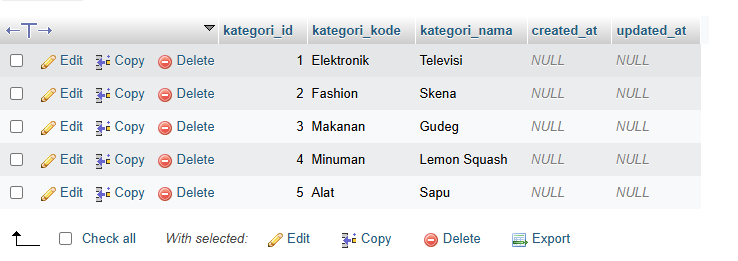
1. Ok, data seeder berhasil di masukkan ke database.
2. Sekarang coba kalian masukkan data *seeder* untuk table yang lain, dengan ketentuan seperti berikut

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Nama Tabel** | **Jumlah Data** | **Keterangan** |
| 1 | m\_kategori | 5 | 5 kategori barang |
| 2 | m\_barang | 10 | 10 barang yang berbeda |
| 3 | t\_stok | 10 | Stok untuk 10 barang |
| 4 | t\_penjualan | 10 | 10 transaksi penjualan |
| 5 | t\_penjualan\_detail | 30 | 3 barang untuk setiap transaksi penjualan |

1. Jika sudah, laporkan hasil Praktikum-3 ini dan *commit* perubahan pada *git*

**

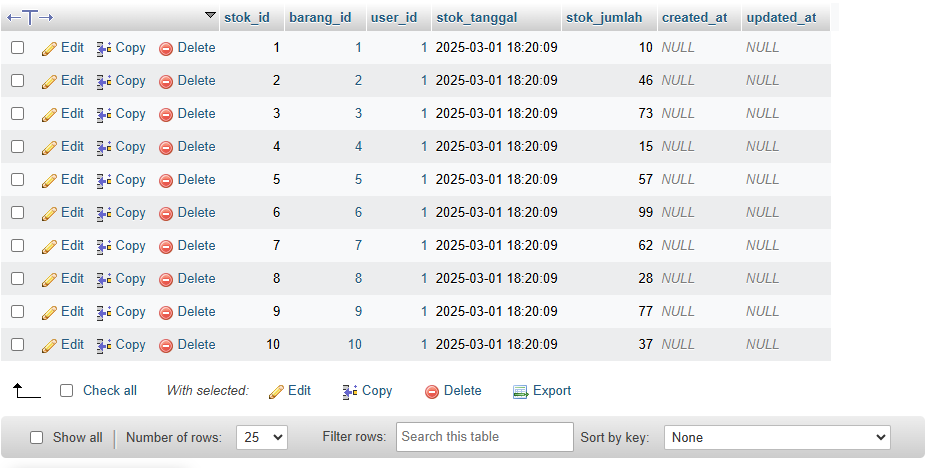
1. **Kategori**

****

1. **Barang**

****

1. **Stok**

****

1. **Penjualan**

****

1. **Penjualan Detail**

****

# DB FACADE

DB Façade merupakan fitur dari Laravel yang digunakan untuk melakukan *query* secara langsung dengan mengetikkan perinta SQL secara utuh (*raw query*). Disebut *raw query* (query mentah) karena penulisan query pada DB Façade langsung ditulis sebagaimana yang biasa dituliskan pada database, seperti “select \* from m\_user” atau “insert into m\_user…” atau “update m\_user set … Where …”

*Raw query* adalah cara paling dasar dan tradisional yang ada di Laravel. Raw query terasa familiar karena biasa kita pakai ketika melakukan query langsung ke database.

**INFO**

Dokumentasi penggunaan DB Façade bisa dicek di laman ini

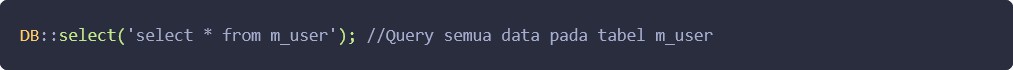
<https://laravel.com/docs/10.x/database#running-queries>

Terdapat banyak method yang bisa digunakan pada DB Façade ini. Akan tetapi yang kita pelajari cukup 4 (empat) method yang umum dipakai, yaitu

1. DB::select()

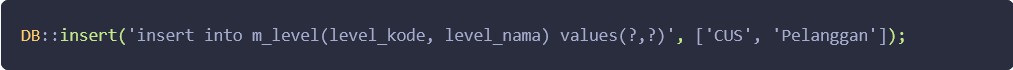
Method ini digunakan untuk mengambil data dari database. Method ini

**mengembalikan *(return)*** data hasil *query*. Contoh



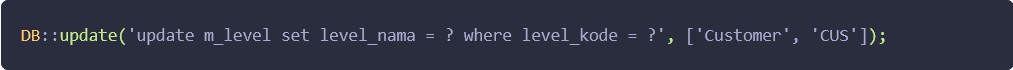
1. DB::insert()

Method ini digunakan untuk memasukkan data pada table database. Method ini **tidak memiliki nilai pengembalian *(no return)***. Contoh



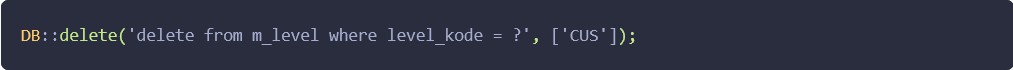
1. DB::update()

Method ini digunakan saat menjalankan *raw query* untuk meng-update data pada database. Method ini **memiliki nilai pengembalian *(return)*** berupa jumlah baris data yang ter-*update*. Contoh



1. DB::delete()

Method ini digunakan saat menjalankan *raw query* untuk menghapus data dari table. Method ini **memiliki nilai pengembalian *(return)*** berupa jumlah baris data yang telah dihapus. Contoh



Ok, sekarang mari kita coba praktikkan menggunakan DB Façade pada project kita

**Praktikum 4** – Implementasi DB Facade

1. Kita buat controller dahulu untuk mengelola data pada table m\_level



1. Kita modifikasi dulu untuk *routing*-nya, ada di PWL\_POS/routes/web.php

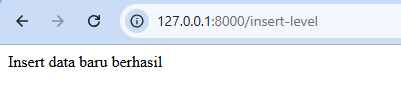


1. Selanjutnya, kita modifikasi file LevelController untuk menambahkan 1 data ke table

m\_level



1. Kita coba jalankan di browser dengan url localhost/PWL\_POS/public/level dan amati apa yang terjadi pada table m\_level di database, *screenshot* perubahan yang ada pada table m\_level





1. Selanjutnya, kita modifikasi lagi file LevelController untuk meng-*update* data di table

m\_level seperti berikut

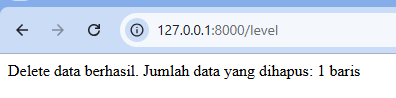


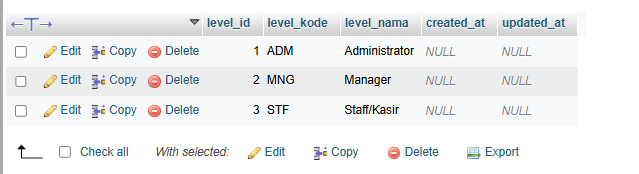
1. Kita coba jalankan di browser dengan url localhost/PWL\_POS/public/level lagi dan amati apa yang terjadi pada table m\_level di database, *screenshot* perubahan yang ada pada table m\_level



1. Kita coba modifikasi lagi file LevelController untuk melakukan proses hapus data

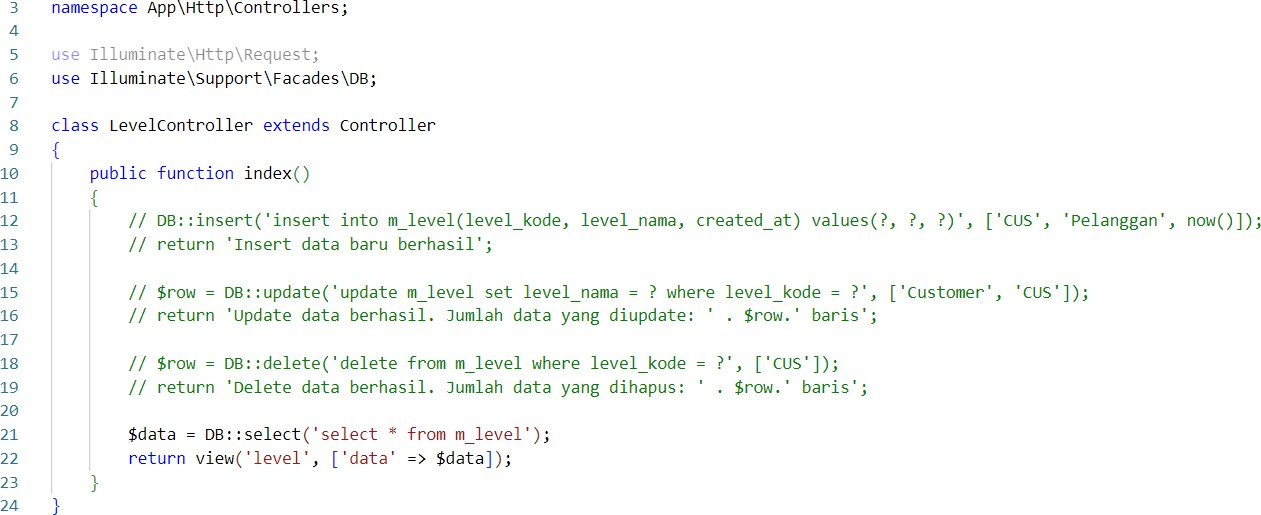




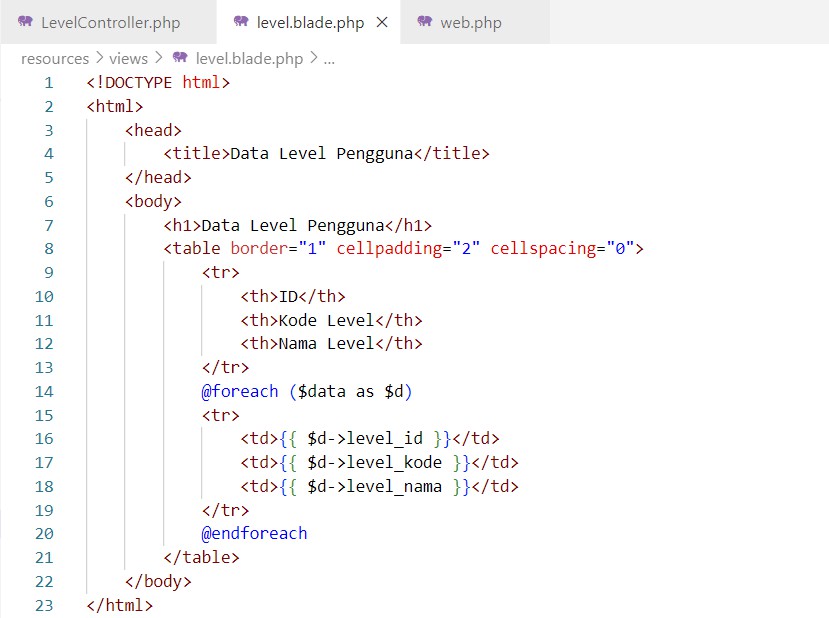


1. Method terakhir yang kita coba adalah untuk menampilkan data yang ada di table

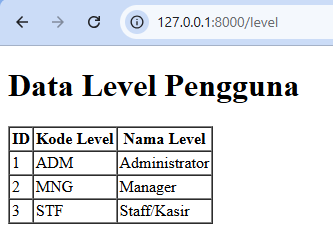
m\_level. Kita modifikasi file LevelController seperti berikut



1. Coba kita perhatikan kode yang diberi tanda kotak merah, berhubung kode tersebut memanggil view(‘level’), maka kita buat file view pada VSCode di PWL\_POS/resources/view/level.blade.php



1. Silahkan dicoba pada browser dan amati apa yang terjadi

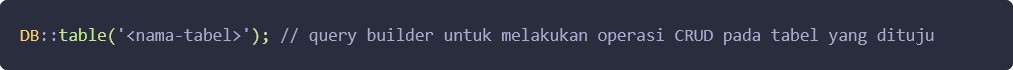


1. Laporkan hasil Praktikum-4 ini dan *commit* perubahan pada *git*.

# QUERY BUILDER

*Query builder* adalah fitur yang disediakan Laravel untuk melakukan proses CRUD *(create, retrieve/read, update, delete)* pada database. Berbeda dengan *raw query* pada DB Facede yang mengharuskan kita menulis perintah SQL, pada *query builder* perintah SQL ini diakses menggunakan method. Jadi, kita tidak menulis perintah SQL secara langsung, melainkan cukup memanggil method-method yang ada di *query builder*.

Query builder membuat kode kita menjadi rapi dan lebih mudah dibaca. Selain itu *query builder* tidak terikat ke satu jenis database, jadi query builder bisa digunakan untuk mengakses berbagai jenis database seperti MySQL, MariaDB, PostgreSQL, SQL Server, dll. Jika suatu saat ingin beralih dari database MySQL ke PostgreSQL, tidak akan banyak kendala. Namun kelemahan dari *query builder* adalah kita harus mengetahui method-method apa saja yang ada di *query builder*.



**INFO**

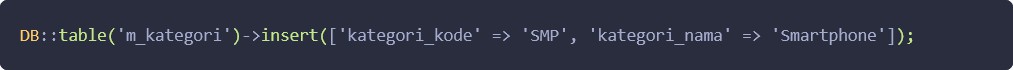
Dokumentasi penggunaan Query Builder pada Laravel bisa dicek di laman ini

<https://laravel.com/docs/10.x/queries>

Ciri khas *query builder* Laravel adalah kita tentukan dahulu target table yang akan kita akses untuk operasi CRUD.

Perintah pertama yang dilakukan pada query builder adalah menentukan nama table yang akan dilakukan operasi CRUD. Kemudian baru disusul method yang ingin digunakan sesuai dengan peruntukannya. Contoh

1. Perintah untuk *insert* data dengan method insert()



Query yang dihasilkan dari kode di atas adalah

insert into m\_kategori(kategori\_kode, kategori\_nama) values(‘SMP’, ‘Smartphone’);

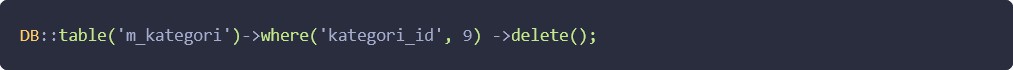
1. Perintah untuk *update* data dengan method where() dan update()



Query yang dihasilkan dari kode di atas adalah

update m\_kategori set kategori\_nama = ‘Makanan Ringan’ where kategori\_id = 1;

1. Perintah untuk *delete* data dengan method where() dan delete()



Query yang dihasilkan dari kode di atas adalah

delete from m\_kategori where kategori\_id = 9;

1. Perintah untuk ambil data

|  |  |
| --- | --- |
| ***Method Query Builder*** | **Query yang dihasilkan** |
| DB::table(‘m\_kategori’)->get(); | select \* from m\_kategori |
| DB::table(‘m\_kategori’)  ->where(‘kategori\_id’, 1)->get(); | select \* from m\_kategori where kategori\_id = 1; |
| DB::table(‘m\_kategori’)  ->select(‘kategori\_kode’)  ->where(‘kategori\_id’, 1)->get(); | select kategori\_kode from m\_kategori where kategori\_id = 1; |

**Praktikum 5** – Implementasi *Query Builder*

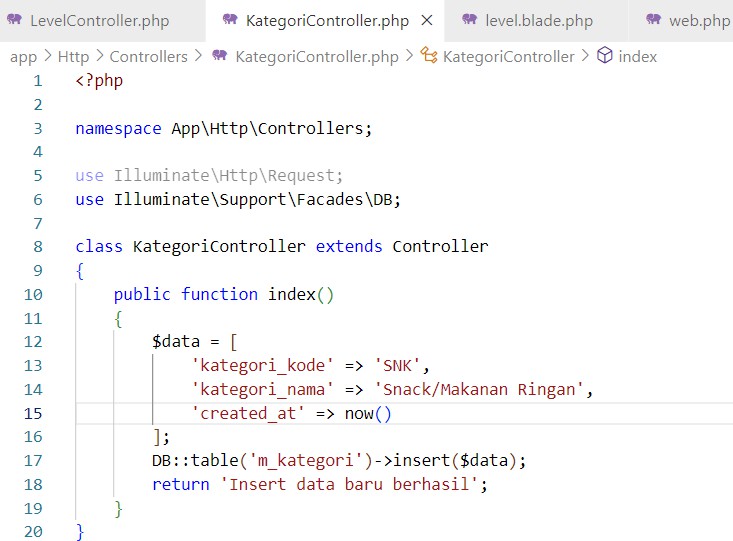
1. Kita buat controller dahuku untuk mengelola data pada table m\_kategori



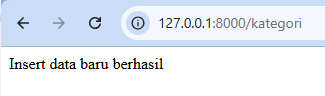
1. Kita modifikasi dulu untuk routing-nya, ada di PWL\_POS/routes/web.php

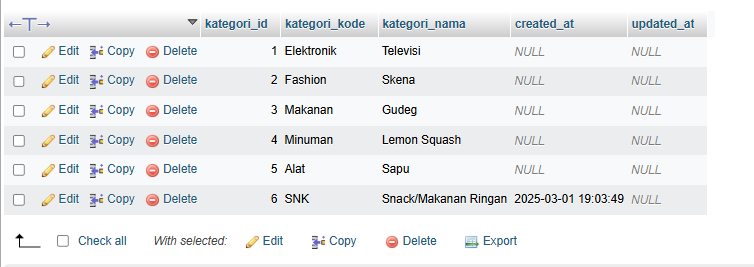


1. Selanjutnya, kita modifikasi file KategoriController untuk menambahkan 1 data ke table m\_kategori



1. Kita coba jalankan di browser dengan url localhost/PWL\_POS/public/kategori dan amati apa yang terjadi pada table m\_kategori di database, *screenshot* perubahan yang ada pada table m\_kategori

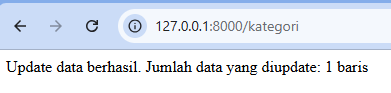


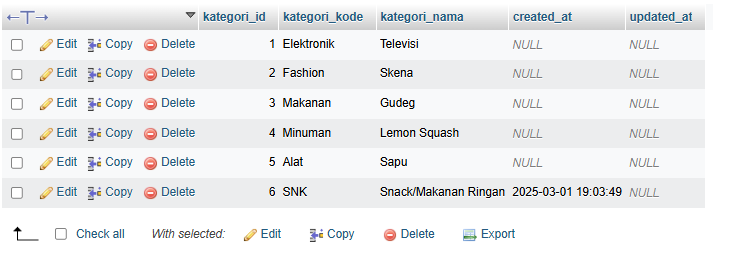


1. Selanjutnya, kita modifikasi lagi file KategoriController untuk meng-*update* data di table m\_kategori seperti berikut

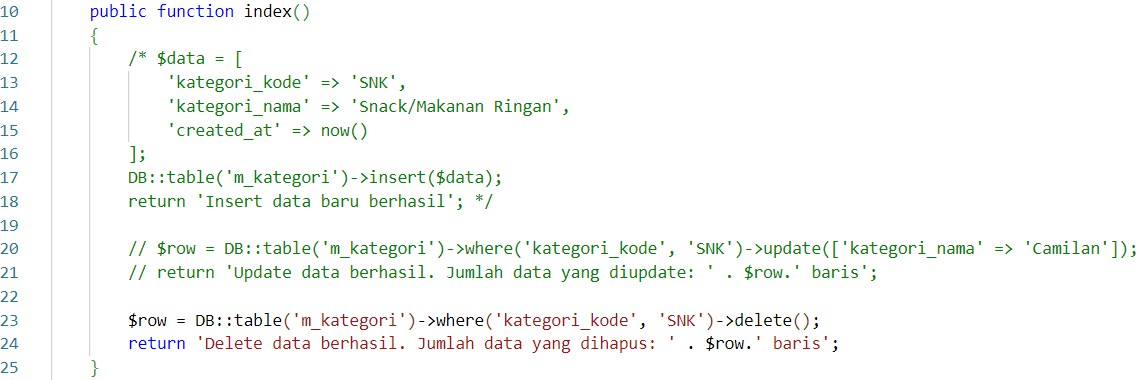


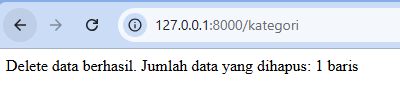
1. Kita coba jalankan di browser dengan url localhost/PWL\_POS/public/kategori lagi dan amati apa yang terjadi pada table m\_kategori di database, *screenshot* perubahan yang ada pada table m\_kategori

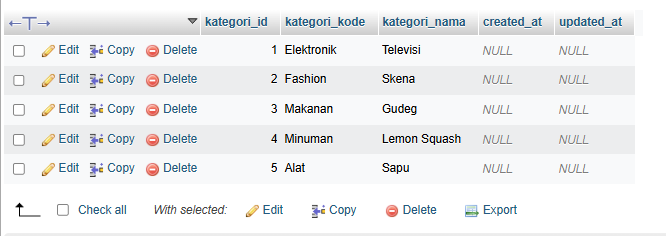




1. Kita coba modifikasi lagi file KategoriController untuk melakukan proses hapus data

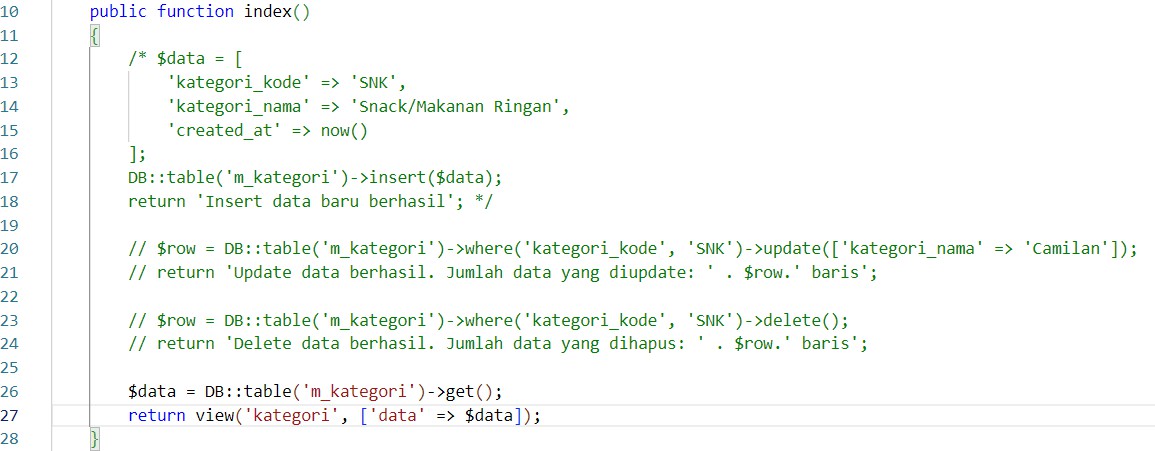




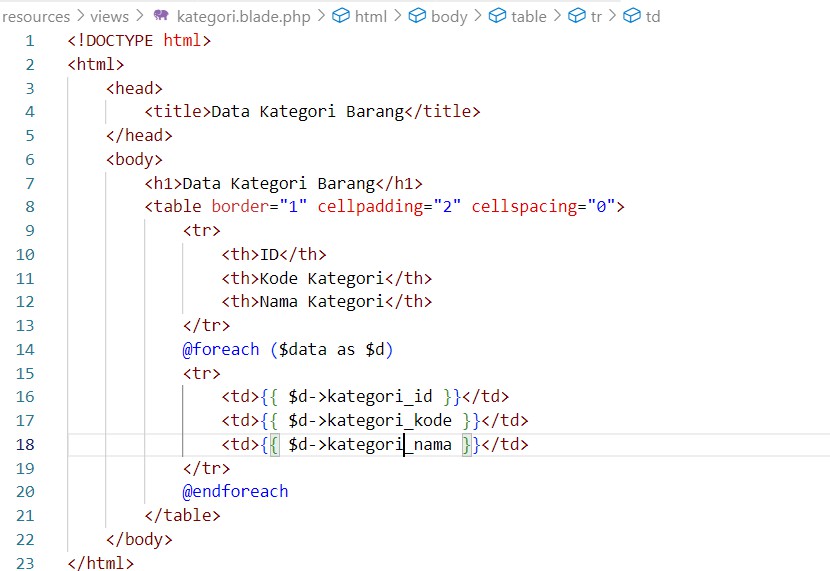


1. Method terakhir yang kita coba adalah untuk menampilkan data yang ada di table

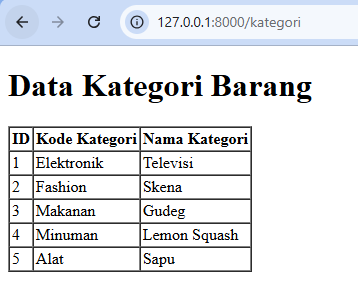
m\_kategori. Kita modifikasi file KategoriController seperti berikut



1. Coba kita perhatikan kode yang diberi tanda kotak merah, berhubung kode tersebut memanggil view(‘kategori’), maka kita buat file view pada VSCode di PWL\_POS/resources/view/kategori.blade.php



1. Silahkan dicoba pada browser dan amati apa yang terjadi.



1. Laporkan hasil Praktikum-5 ini dan *commit* perubahan pada *git*

# ELOQUENT ORM

Eloquent ORM adalah fitur bawaan dari laravel. Eloquent ORM adalah cara pengaksesan database dimana setiap baris tabel dianggap sebagai sebuah object. Kata ORM sendiri merupakan singkatan dari ***Object-relational mapping***, yakni suatu teknik programming untuk mengkonversi data ke dalam bentuk object.



**INFO**

Eloquent ORM memerlukan Model untuk proses konversi data pada tabel menjadi object. Object inilah yang nantinya akan kita akses dari dalam controller. Oleh karena itu **membuat**

**Model pada Laravel berarti menggunakan Eloquent ORM**. Silahkan cek disini

<https://laravel.com/docs/10.x/eloquent>

Perintah untuk membuat model adalah sebagai berikut

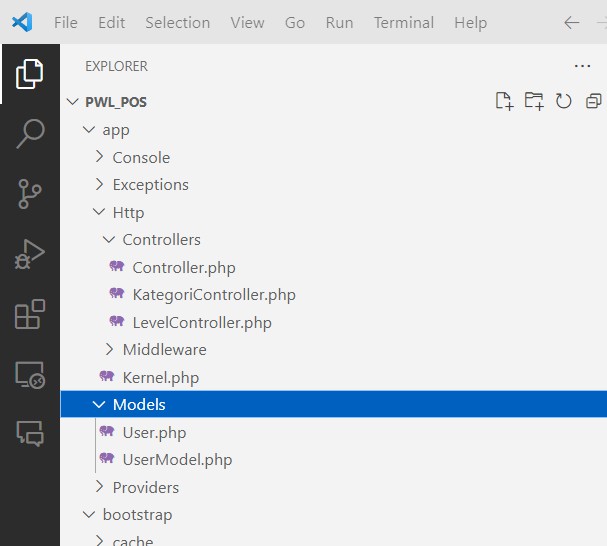
Untuk bisa melakukan operasi ***CRUD*** (*create, read/retrieve, update, delete*), kita harus membuat sebuah model sesuai dengan target tabel yang ingin digunakan. Jadi,

**dalam 1 model, merepresentasikan 1 tabel database.**

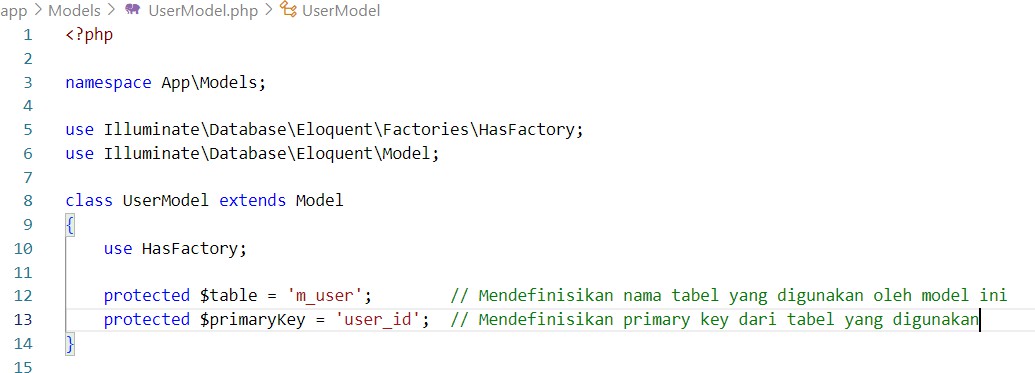
**Praktikum 6** – Implementasi Eloquent ORM

* 1. Kita buat file model untuk tabel m\_user dengan mengetikkan perintah





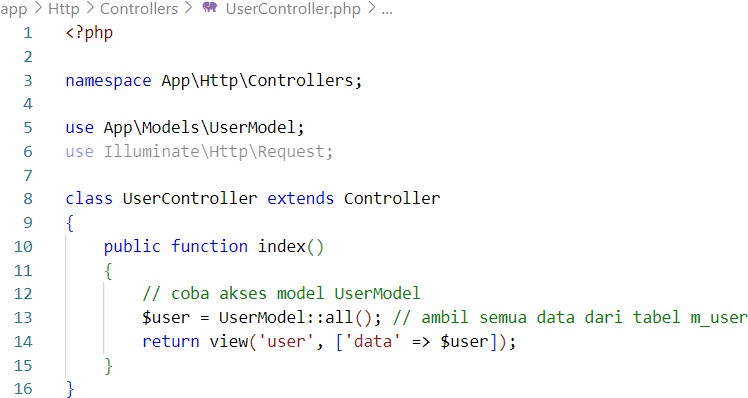
* 1. Setelah berhasil generate model, terdapat 2 file pada folder model yaitu file User.php bawaan dari laravel dan file UserModel.php yang telah kita buat. Kali ini kita akan menggunakan file UserModel.php
  2. Kita buka file UserModel.php dan modifikasi seperti berikut



* 1. Kita modifikasi route web.php untuk mencoba routing ke controller UserController



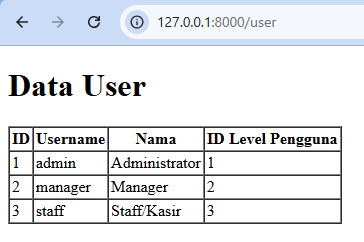
* 1. Sekarang, kita buat file controller UserController dan memodifikasinya seperti berikut



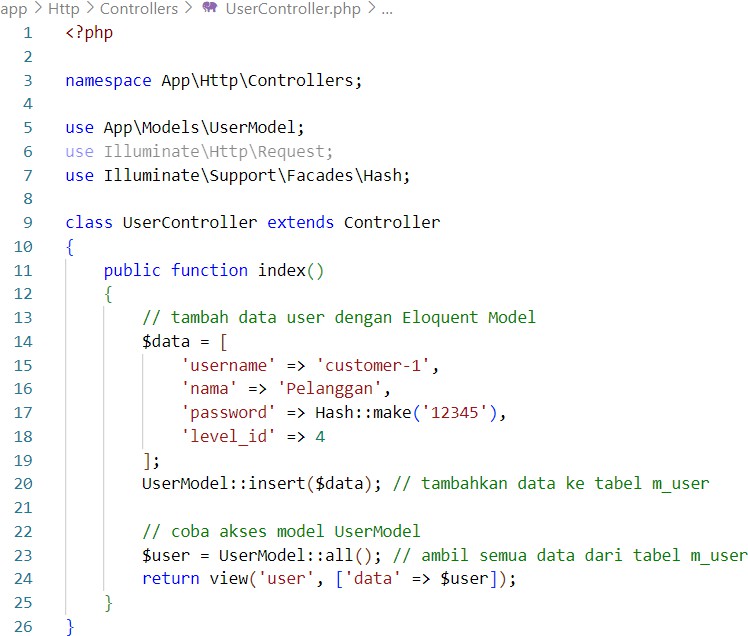
* 1. Kemudian kita buat view user.blade.php



* 1. Jalankan di browser, catat dan laporkan apa yang terjadi



* 1. Setelah itu, kita modifikasi lagi file UserController

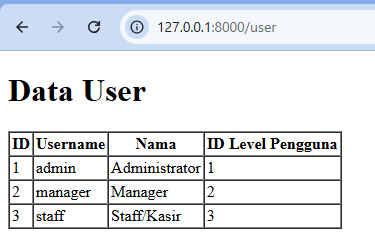


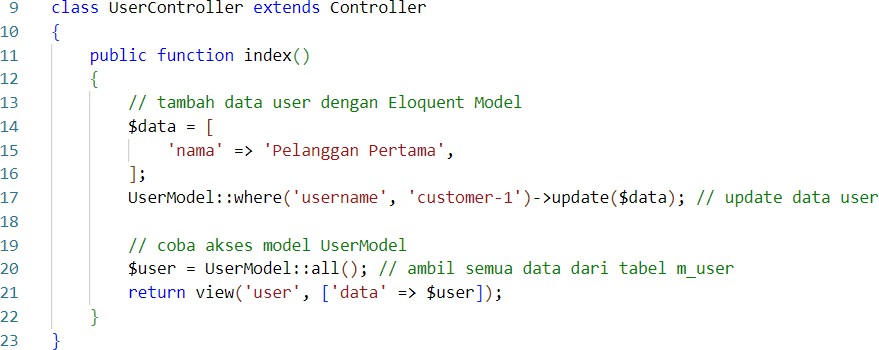
* 1. Jalankan di browser, amati dan laporkan apa yang terjadi

= terjadi error karena m\_user memiliki foreign key milik level\_id yang merujuk pada m\_level pada kolom level\_id, dan kesalahan terjadi karena nilai level\_id ingin memasukkan nilai (4) namun tidak ada dalam tabel m\_level. Solusinya adalah mengubah di sql nya



* 1. Kita modifikasi lagi file UserController menjadi seperti berikut





* 1. Jalankan di browser, amati dan laporkan apa yang terjadi
  2. Jika sudah, laporkan hasil Praktikum-6 ini dan *commit* perubahan pada *git*

# Penutup

Jawablah pertanyaan berikut sesuai pemahaman materi di atas

* 1. **Pada Praktikum 1 - Tahap 5, apakah fungsi dari APP\_KEY pada *file setting* .env Laravel?**

= APP\_KEY adalah kunci enkripsi yang berfungsi sebagai keamanan. Kunci ini berguna untuk menjaga keamanan aplikasi yang menggunakan Laravel Encryption

* 1. **Pada Praktikum 1, bagaimana kita men-*generate* nilai untuk APP\_KEY?**

= menggunakan perintah “php artisan key:generate” untuk menghasilkan APP\_KEY baru dengan menggantikan nilai APP\_KEY sebelumnya

* 1. **Pada Praktikum 2.1 - Tahap 1, secara *default* Laravel memiliki berapa file migrasi? dan untuk apa saja file migrasi tersebut?**

= ada 3 file migration :

1. 2024\_XX\_XX\_000000\_create\_users\_table.php = Membuat tabel users untuk menyimpan data pengguna.
2. **2024\_XX\_XX\_000001\_create\_password\_reset\_tokens\_table.php** = Membuat tabel password\_reset\_tokens untuk menyimpan token reset password.
3. **2024\_XX\_XX\_000002\_create\_failed\_jobs\_table.php** = Membuat tabel failed\_jobs untuk mencatat pekerjaan (jobs) yang gagal dalam queue system.
   1. **Secara *default*, file migrasi terdapat kode $table->timestamps();, apa tujuan/*output* dari fungsi tersebut?**

= $table->timestamps(); secara otomatis akan menambahkan 2 kolom tabel baru yakni created\_at (menyimpan waktu pertama kali dibuat) dan updated\_at (menyimpan waktu terakhir kali dibuat)

* 1. **Pada File Migrasi, terdapat fungsi $table->id(); Tipe data apa yang dihasilkan dari fungsi tersebut?**

= fungsinya adalah menghasilkan kolom yang bertipe data BIGINT yang otomatis menjadi Primary Key dan fitur Auto Increament

* 1. **Apa bedanya hasil migrasi pada table m\_level, antara menggunakan $table->id();**

**dengan menggunakan $table->id('level\_id'); ?**

= perbedaannya adalah

* Jika menggunakan $table->id();, maka nama kolom primary key akan default **id**.
* Jika menggunakan $table->id();, maka nama kolom primary key akan default **id**.
  1. **Pada migration, Fungsi ->unique() digunakan untuk apa?**

= Fungsi ->unique() digunakan untuk menetapkan bahwa nilai dalam suatu kolom harus **unik yang artinya tidak boleh ada data yang dobel**

* 1. **Pada Praktikum 2.2 - Tahap 2, kenapa kolom level\_id pada tabel m\_user menggunakan $tabel->unsignedBigInteger('level\_id'), sedangkan kolom level\_id pada tabel m\_level menggunakan $tabel->id('level\_id') ?**

= jika $table->id('level\_id'); Membuat kolom primary key di tabel m\_level. Kemudian $table->unsignedBigInteger('level\_id'); Membuat kolom foreign key di tabel m\_user agar bisa merujuk ke primary key di m\_level.

* 1. **Pada Praktikum 3 - Tahap 6, apa tujuan dari Class Hash? dan apa maksud dari kode program Hash::make('1234');?**

= Hash::make('1234'); digunakan untuk menyimpan password dalam bentuk terenkripsi agar lebih aman dibandingkan menyimpannya secara langsung dalam teks biasa.

* 1. **Pada Praktikum 4 - Tahap 3/5/7, pada *query builder* terdapat tanda tanya (?), apa kegunaan dari tanda tanya (?) tersebut?**

= fungsinya adalah :

* **Mencegah SQL Injection** dengan memastikan nilai dimasukkan secara aman.
* **Meningkatkan kinerja** karena query dapat dikompilasi sekali dan digunakan kembali dengan nilai yang berbeda.
  1. **Pada Praktikum 6 - Tahap 3, apa tujuan penulisan kode protected $table = ‘m\_user’; dan protected $primaryKey = ‘user\_id’; ?**

= berikut tujuannya :

1. protected $table = 'm\_user'; menentukan jika model tersebut berhubungan dengan tabel m\_user.
2. protected $primaryKey = 'user\_id'; menentukan bahwa kolom primary key pada tabel m\_user adalah user\_id. Secara default, Laravel menganggap primary key bernama id, jadi jika menggunakan nama lain, harus dideklarasikan secara eksplisit.
   1. **Menurut kalian, lebih mudah menggunakan mana dalam melakukan operasi CRUD ke database (*DB Façade / Query Builder / Eloquent ORM*) ? jelaskan**

= menurut saya lebih mudah menggunakan antara *Query Builder* ***dan*** *Eloquent ORM karena penggunaan Query Builder lebih dipahami strukturnya*

*\*\*\* Sekian, dan selamat belajar \*\*\**