KEMENTERIAN PENDIDIKAN, KEBUDAYAAN, RISET, DAN TEKNOLOGI
POLITEKNIK NEGERI MALANG
JURUSAN TEKNOLOGI INFORMASI
Jl. Soekarno Hatta No. 9, Jatimulyo, Lowokwaru, Malang 65141
Telp. (0341) 404424 – 404425, Fax (0341) 404420
http://www.polinema.ac.id

x

Mata Kuliah : Pemrograman Web Lanjut (PWL)
Program Studi : D4 – Teknik Informatika / D4 – Sistem Informasi Bisnis
Semester : 4 (empat) / 6 (enam)
Pertemuan ke- : 1 (satu)

Nama : M. Fairuz Daffa A.
Kelas : SIB 2B / 15
NIM : 2341760079

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

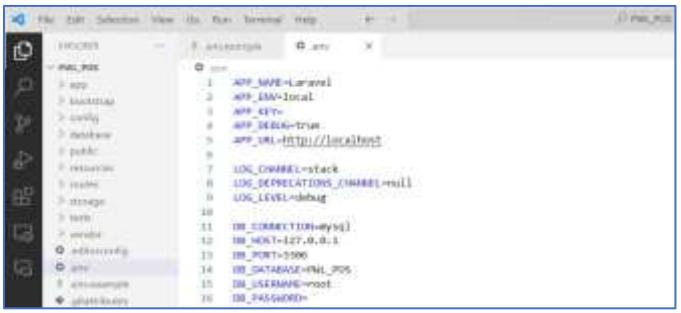
A. PENGATURAN DATABASE

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.

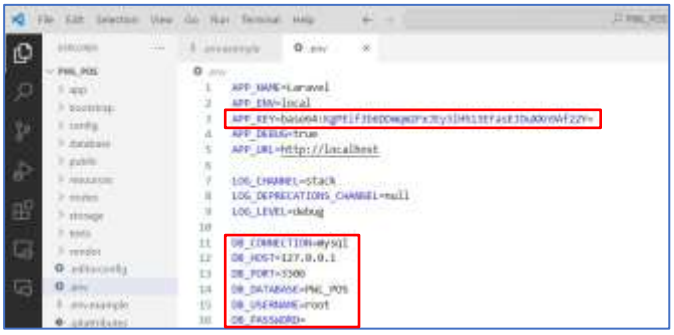
Praktikum 1 - pengaturan database:

1. Buka aplikasi phpMyAdmin, dan buat database baru dengan nama `PWL_POS`

2. Buka aplikasi VSCode dan buka folder project `PWL_POS` yang sudah kita buat
3. Copy file `.env.example` menjadi `.env`
4. Buka file `.env`, dan pastikan konfigurasi **`APP_KEY`** bernilai. Jika belum bernilai silahkan kalian *generate* menggunakan `php artisan`.



5. Edit file `.env` dan sesuaikan dengan database yang telah dibuat



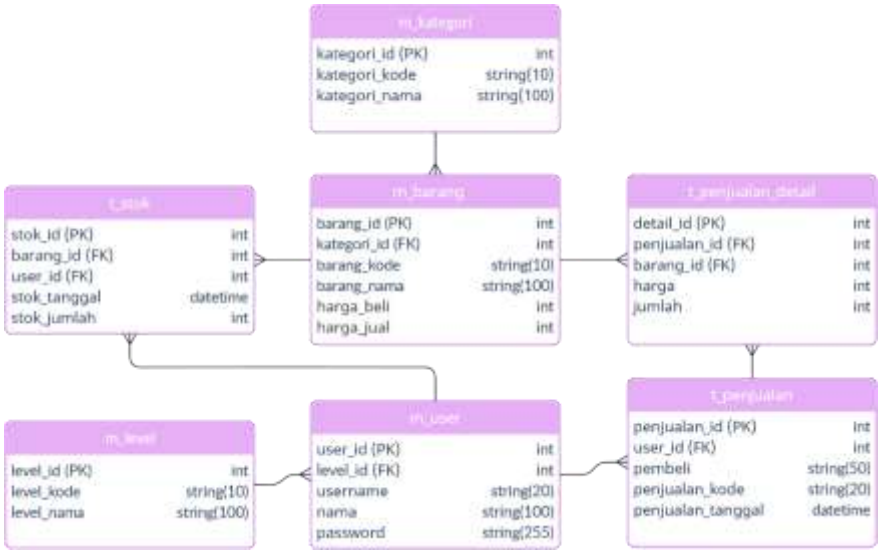
6. Laporkan hasil Praktikum-1 ini dan *commit* perubahan pada *git*.

B. 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

a. Menggunakan `artisan` untuk membuat *file migration*

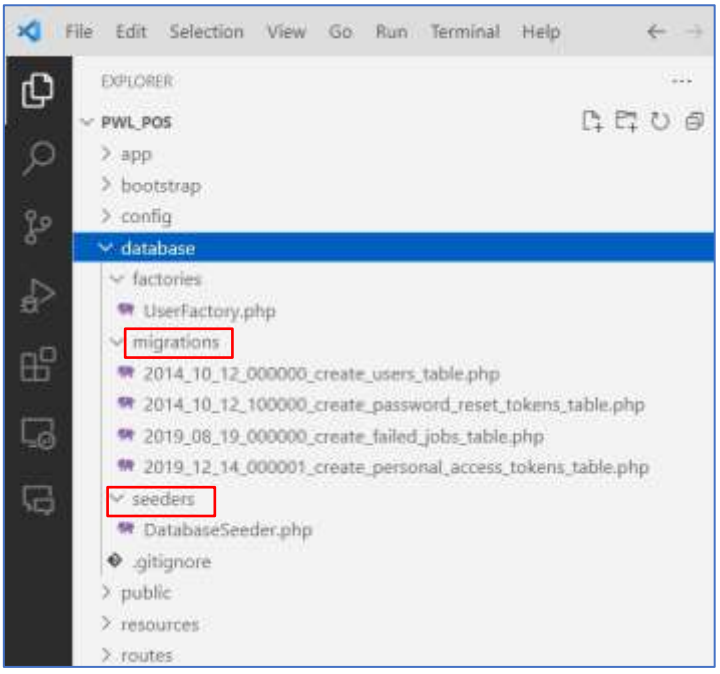
```
php artisan make:migration <nama-file-tabel> --create=<nama-tabel>
```

b. Menggunakan `artisan` untuk membuat *file model + file migration*

```
php artisan make:model <nama-model> -m
```

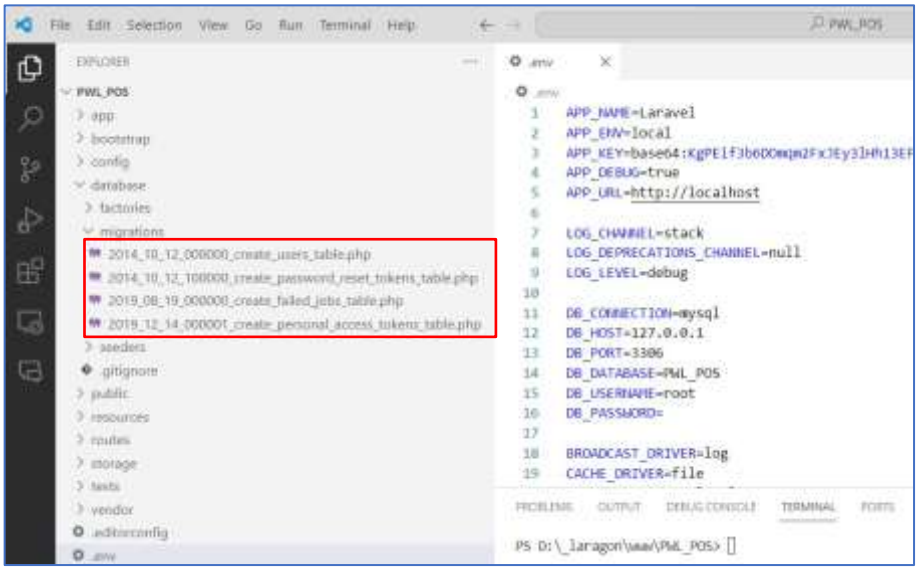
Perintah **`-m`** di atas adalah *shorthand* untuk opsi membuat file migrasi berdasarkan model yang dibuat.

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



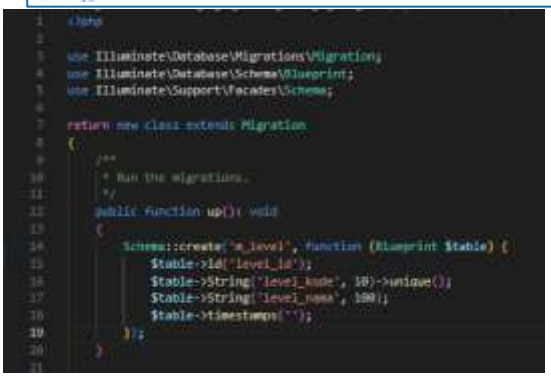
2. Kita abaikan dulu yang di kotak merah (jangan di hapus)
3. Kita buat file migrasi untuk table `m_level` dengan perintah

```
php artisan make:migration create_m_level_table --create=m_level
```

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('m_level', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('m_level');
    }
};
```

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

```php
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('m_level', function (Blueprint $table) {
            $table->id('level_id');
            $table->string('level_kode', 10)->unique();
            $table->string('level_nama', 100);
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('m_level');
    }
};
```



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

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

```
php artisan migrate
```

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

7. Ok, table sudah dibuat di database
8. Buat table *database* dengan *migration* untuk table `m_kategori` yang sama-sama tidak memiliki *foreign key*

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('m_kategori', function (Blueprint $table) {
            $table->unsignedBigInteger('kategori_id', 20);
            $table->String('kategori_kode', 10);
            $table->String('kategori_nama', 100);
            $table->timestamp('create_at');
            $table->timestamp('updated_at');
        });
    }

```

9. 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`

```
php artisan make:migration create_m_user_table --table=m_user
```

2. Buka file migrasi untuk table `m_user`, dan modifikasi seperti berikut

```php
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('m_user', function (Blueprint $table) {
            $table->id('user_id');
            $table->unsignedBigInteger('level_id')->index(); // indexing untuk ForeignKey
            $table->string('username', 20)->unique(); // unique untuk memastikan tidak ada username yang sama
            $table->string('nama', 100);
            $table->string('password');
            $table->timestamps();

            // Mendefinisikan Foreign Key pada kolom level_id mengacu pada kolom level_id di tabel m_level
            $table->foreign('level_id')->references('level_id')->on('m_level');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('m_user');
    }
};
```

database > migrations > 2025_02_28_014416_create_m_user_table.php > class > up > Closure

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('m_user', function (Blueprint $table) {
            $table->id('user_id');
            $table->unsignedBigInteger('level_id')->index();
            $table->String('username', 20)->unique();
            $table->String('nama', 100);
            $table->String('password');
            $table->timestamps('');

            $table->foreign('level_id')->references('level_id')->on('m_level');
        });
    }

    /**
```

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

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

m_barang
t_penjualan
t_stok
t_penjualan_detail

m_barang

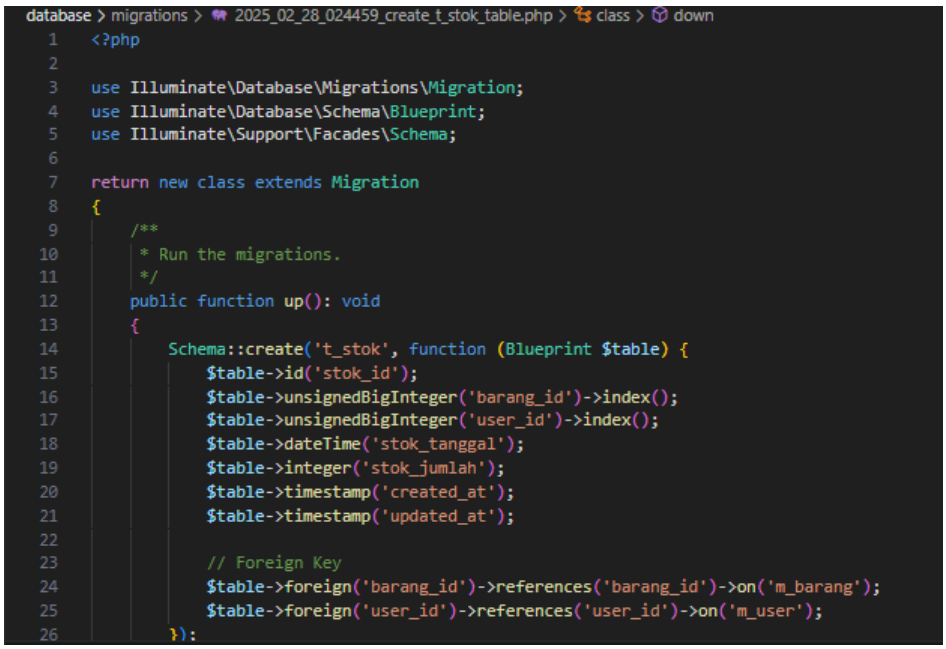
```php
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('m_barang', function (Blueprint $table) {
            $table->id('barang_id');
            $table->unsignedBigInteger('kategori_id')->index();
            $table->string('barang_kode', 10);
            $table->string('barang_nama', 100);
            $table->integer('harga_beli');
            $table->integer('harga_jual');
            $table->timestamp('created_at');
            $table->timestamp('updated_at');

            // Foreign Key
            $table->foreign('kategori_id')->references('kategori_id')->on('m_kategori');
        });
    }
```

t_penjulan

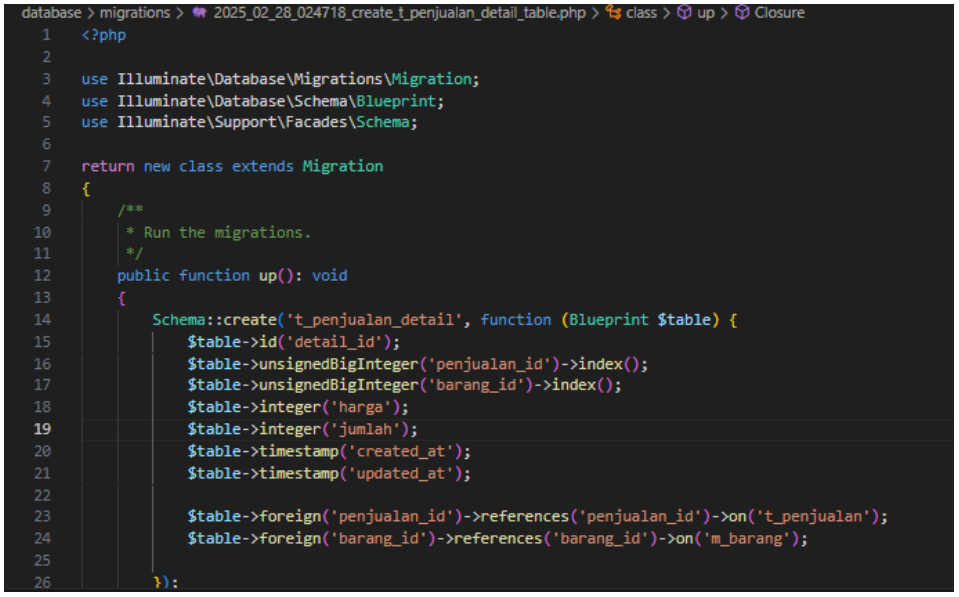
database > migrations > 2025_02_28_024634_create_t_penjualan_table.php > class > down

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('t_penjualan', function (Blueprint $table) {
            $table->id('penjualan_id');
            $table->unsignedBigInteger('user_id')->index();
            $table->string('pembeli');
            $table->string('penjualan_kode');
            $table->dateTime('penjualan_tanggal');
            $table->timestamp('created_at');
            $table->timestamp('updated_at');

            // Foreign Key
            $table->foreign('user_id')->references('user_id')->on('m_user');
        });
```

t_stok

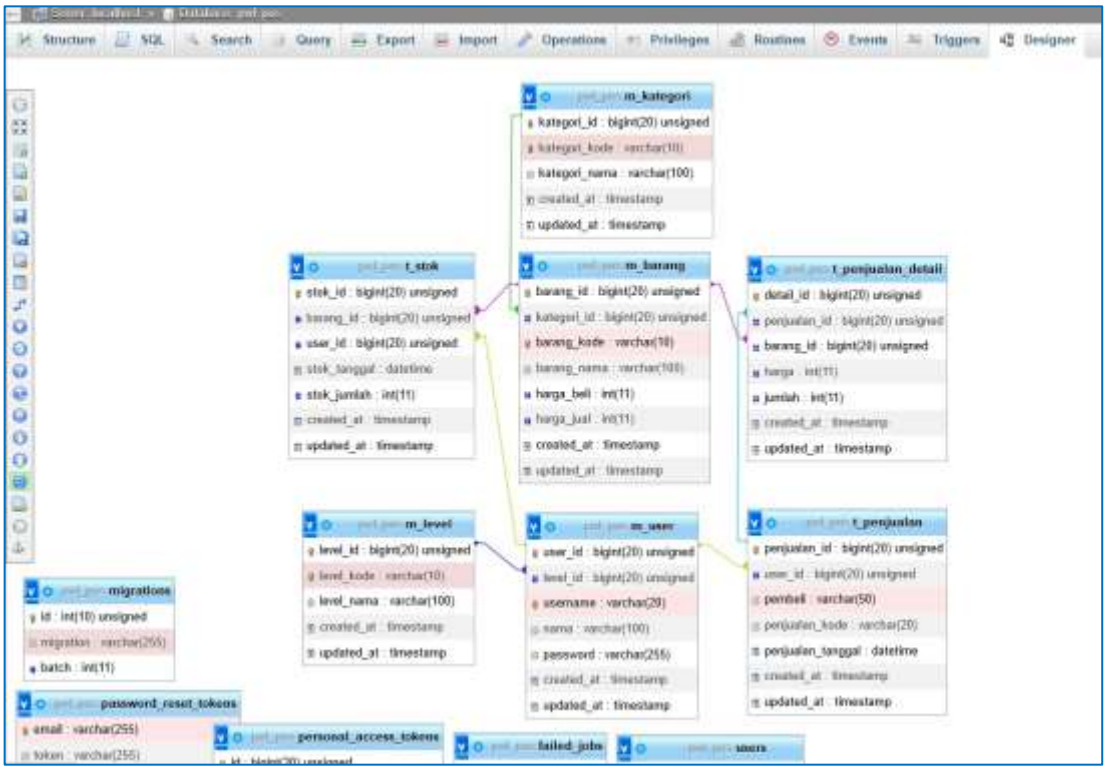


t_penjualan_detail

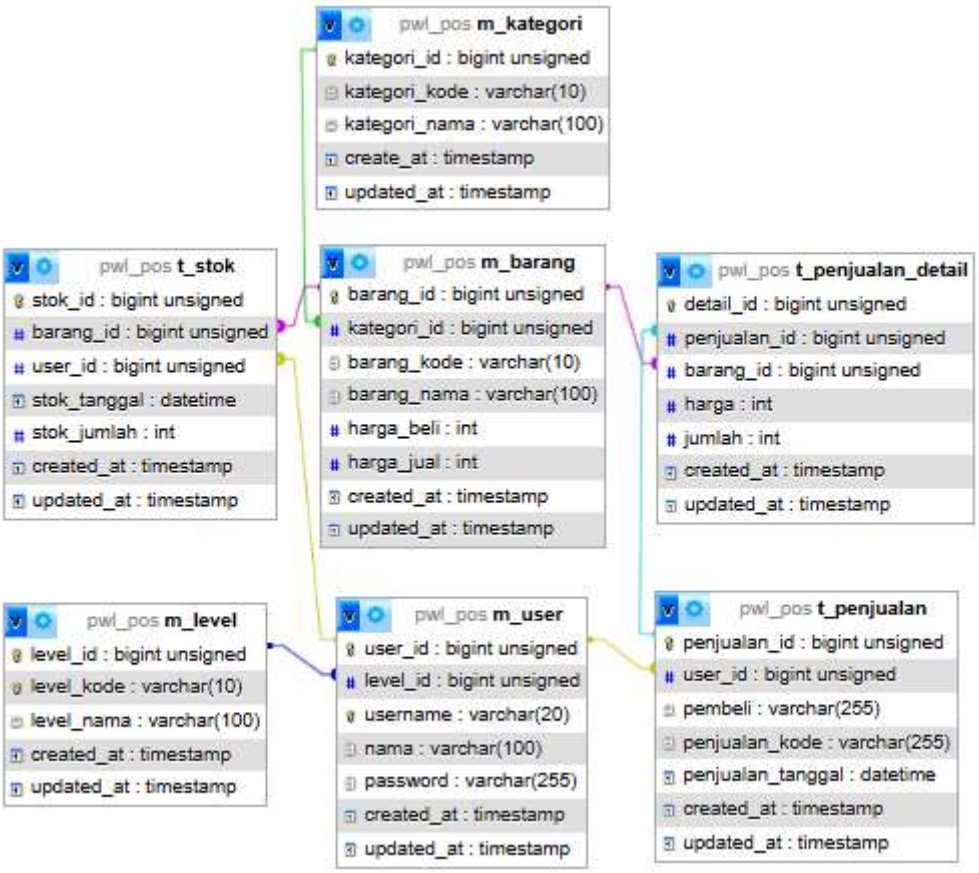


4. Jika semua file migrasi sudah di buat dan dijalankan maka bisa kita lihat tampilan

designer pada **phpMyAdmin** seperti berikut



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



C. 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

```
php artisan make:seeder <nama-class-seeder>
```

   Perintah tersebut akan men-generate file seeder pada folder `PWL_POS/database/seeders`

2. Dan perintah untuk **menjalankan *`file seeder`*** seperti berikut

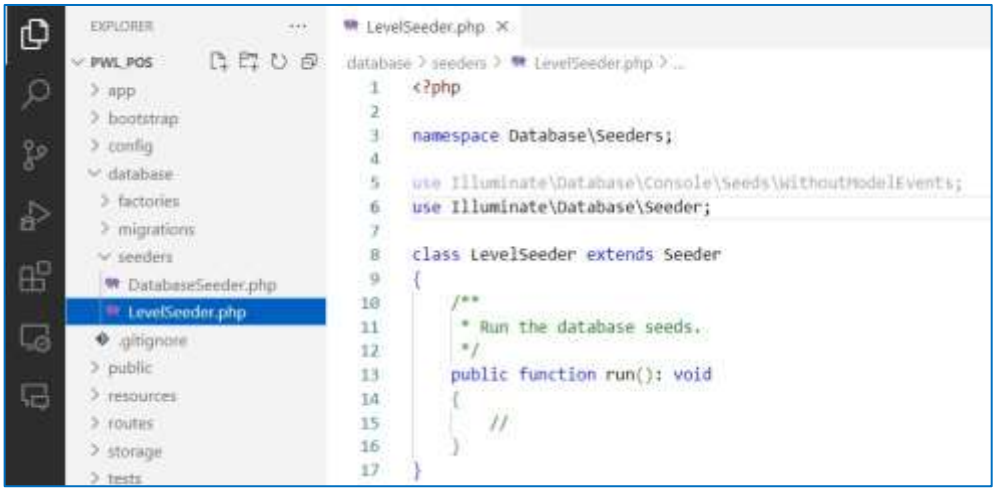
```
php artisan db:seed --class=<nama-class-seeder>
```

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

```
php artisan make:seeder LevelSeeder
```



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

```
database > seeders > LevelSeeder.php > ...
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;

class LevelSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        $data = [
            ['level_kode' => 'ADM', 'level_nama' => 'Administrator'],
            ['level_kode' => 'MNG', 'level_nama' => 'Manager'],
            ['level_kode' => 'STF', 'level_nama' => 'Staff/Kasir'],
        ];

        // Gunakan upsert untuk mencegah duplikasi data
        DB::table('m_level')->upsert($data, ['level_code'], ['level_nama']);
    }
}
```

3. Selanjutnya, kita jalankan file *seeder* untuk table `m_level` pada terminal

```
php artisan db:seed --class=LevelSeeder
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS D:\_laragon\www\PWL_POS> php artisan db:seed --class=LevelSeeder

  INFO  Seeding database.

PS D:\_laragon\www\PWL_POS>
```

4. Ketika *seeder* berhasil dijalankan maka akan tampil data pada table `m_level`

	level_id	level_kode	level_nama	created_at	updated_at
Edit Copy Delete	1	ADM	Administrator	NULL	NULL
Edit Copy Delete	2	MNG	Manager	NULL	NULL
Edit Copy Delete	3	STF	Staff/Kasir	NULL	NULL

Check all With selected: Edit Copy Delete Export

	level_id	level_kode	level_nama	created_at	updated_at
Edit Copy Delete	1	ADM	Administrator	NULL	NULL
Edit Copy Delete	2	MNG	Manager	NULL	NULL
Edit Copy Delete	3	STF	Staff/Kasir	NULL	NULL

5. Sekarang kita buat file *seeder* untuk table `m_user` yang me-*refer* ke table `m_level`

```
php artisan make:seeder UserSeeder
```

6. Modifikasi file `class UserSeeder` seperti berikut

```php
class UserSeeder extends Seeder
{
    public function run(): void
    {
        $data = [
            [
                'user_id' => 1,
                'level_id' => 1,
                'username' => 'admin',
                'nama' => 'Administrator',
                'password' => Hash::make('12345'), // class untuk mengenkripsi/hash password
            ],
            [
                'user_id' => 2,
                'level_id' => 2,
                'username' => 'manager',
                'nama' => 'Manager',
                'password' => Hash::make('12345'),
            ],
            [
                'user_id' => 3,
                'level_id' => 3,
                'username' => 'staff',
                'nama' => 'Staff/Kasir',
                'password' => Hash::make('12345'),
            ],
        ];
        DB::table('m_user')->insert($data);
    }
}
```

database > seeders > UserSeeder.php > UserSeeder > run

```php
<?php

namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Hash;

class UserSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        $data = [
            [
                'user_id' => 1,
                'level_id' => 1,
                'username' => 'admin',
                'nama' => 'Administrator',
                'password' => Hash::make('12345'),
            ],
            [
                'user_id' => 2,
                'level_id' => 2,
                'username' => 'manager',
                'nama' => 'Manager',
                'password' => Hash::make('12345'),
            ],
            [
                'user_id' => 3,
                'level_id' => 3,
                'username' => 'staff',
                'nama' => 'Staff/Kasir',
                'password' => Hash::make('12345'),
            ],
```

7. Jalankan perintah untuk mengeksekusi class `UserSeeder`

```
php artisan db:seed --class=UserSeeder
```

8. Perhatikan hasil seeder pada table `m_user`

	user_id	level_id	username	nama	password
Edit Copy Delete	1	1	admin	Administrator	$2y$12$Tevu4dDO1CUAQpeM6H.Vp.LySwhY.4oAKU7FzwS6fXV...
Edit Copy Delete	2	2	manager	Manager	$2y$12$Ajfns20/FdPTeUgghz31muEhlFaruLxkh5wvZ9NGRpu...
Edit Copy Delete	3	3	staff	Staff/Kasir	$2y$12$Gi23TqGclW5pYeR0VL4o5OxPwb3Osk99VMy/BHnbJ9W...

Check all With selected: Edit Copy Delete Export

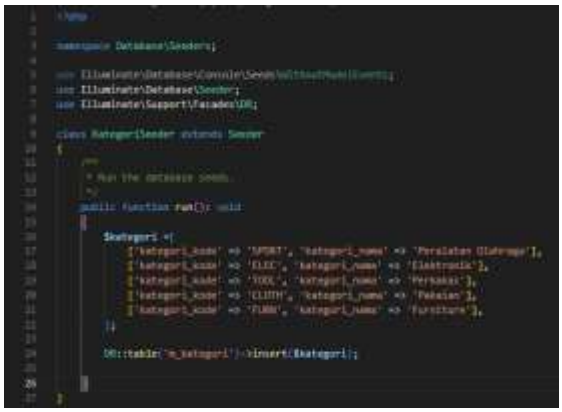
	user_id	level_id	username	nama	password	created_at	updated_at
Edit Copy Delete	1	1	admin	Administrator	$2y$12$7sxl/W1mobfd3XaGPqfXqeqlvAmuR6kHjAdncX1J1g6...	NULL	NULL
Edit Copy Delete	2	2	manager	Manager	$2y$12$3kKPXo5j9zfISCzt5YhUlemBE8LEC7rPqxNfjJLOihp...	NULL	NULL
Edit Copy Delete	3	3	staff	Staff/Kasir	$2y$12$CTStRAgChUUsCOt27MiXe.nhv67Qa6RWunK.hy1uhyC...	NULL	NULL

Check all With selected: Edit Copy Delete Export

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

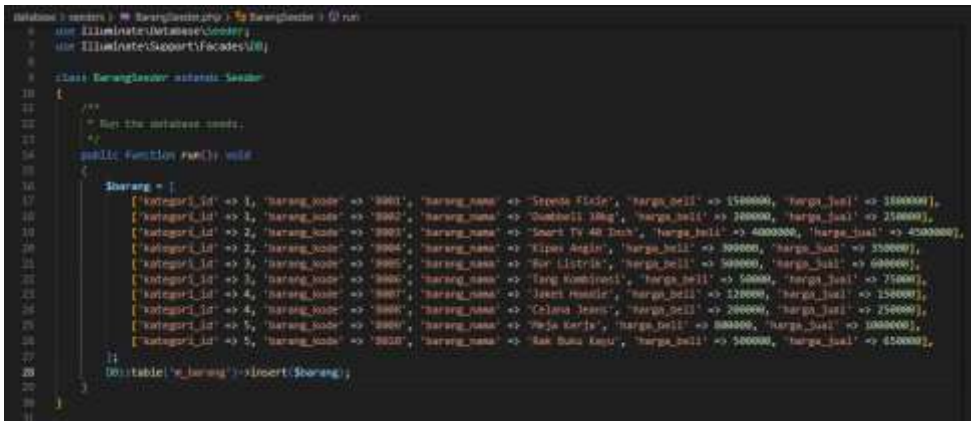
No	Nama Tabel	Jumlah Data	Keterangan
1	`m_kategori`	5	5 kategori barang
2	`m_barang`	10	10 barang yang berbeda
3	`t_stok`	10	Stok untuk 10 barang
4	`t_penjualan`	10	10 transaksi penjualan
5	`t_penjualan_detail`	30	3 barang untuk setiap transaksi penjualan

m_kategori



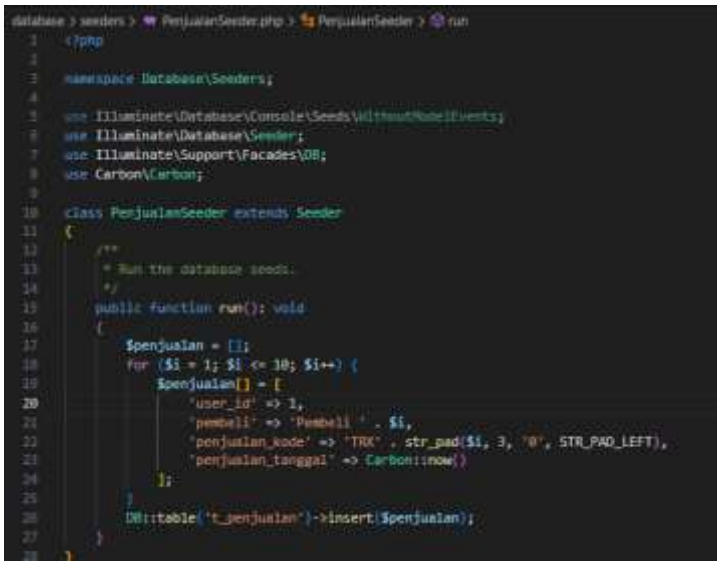
	kategori_id	kategori_kode	kategori_nama	created_at	updated_at
Edit Copy Delete	1	SPORT	Peralatan Olahraga	NULL	NULL
Edit Copy Delete	2	ELEC	Elektronik	NULL	NULL
Edit Copy Delete	3	TOOL	Perkakas	NULL	NULL
Edit Copy Delete	4	CLOTH	Pakaian	NULL	NULL
Edit Copy Delete	5	FURN	Furniture	NULL	NULL

m_barang



	barang_id	kategori_id	barang_kode	barang_nama	harga_beli	harga_jual
Edit Copy Delete	1	1	B001	Sepeda Fixie	1500000	1800000
Edit Copy Delete	2	1	B002	Dumbbell 10kg	200000	250000
Edit Copy Delete	3	2	B003	Smart TV 40 Inch	4000000	4500000
Edit Copy Delete	4	2	B004	Kipas Angin	300000	350000
Edit Copy Delete	5	3	B005	Bor Listrik	500000	600000
Edit Copy Delete	6	3	B006	Tang Kombinasi	50000	75000
Edit Copy Delete	7	4	B007	Jaket Hoodie	120000	150000
Edit Copy Delete	8	4	B008	Celana Jeans	200000	250000
Edit Copy Delete	9	5	B009	Meja Kerja	800000	1000000
Edit Copy Delete	10	5	B010	Rak Buku Kayu	500000	650000

t_stok



	stok_id	barang_id	user_id	stok_tanggal	stok_jumlah
Edit Copy Delete	1	1	1	2025-03-01 03:17:46	10
Edit Copy Delete	2	2	1	2025-03-01 03:17:46	32
Edit Copy Delete	3	3	1	2025-03-01 03:17:46	59
Edit Copy Delete	4	4	1	2025-03-01 03:17:46	92
Edit Copy Delete	5	5	1	2025-03-01 03:17:46	37
Edit Copy Delete	6	6	1	2025-03-01 03:17:46	57
Edit Copy Delete	7	7	1	2025-03-01 03:17:46	61
Edit Copy Delete	8	8	1	2025-03-01 03:17:46	75
Edit Copy Delete	9	9	1	2025-03-01 03:17:46	26
Edit Copy Delete	10	10	1	2025-03-01 03:17:46	89

t_penjualan_detail

```
database > seeders > DetailPenjualanSeeder.php > DetailPenjualanSeeder > run
use Carbon\Carbon;

class DetailPenjualanSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        $detail = [];
        for ($i = 1; $i <= 10; $i++) {
            for ($j = 1; $j <= 3; $j++) {
                $barang_id = rand(1, 10);
                $harga = DB::table('m_barang')->where('barang_id', $barang_id)->value('harga_jual');

                $detail[] = [
                    'penjualan_id' => $i,
                    'barang_id' => $barang_id,
                    'harga' => $harga,
                    'jumlah' => rand(1, 5)
                ];
            }
        }
        DB::table('t_penjualan_detail')->insert($detail);
    }
}
```

	detail_id	penjualan_id	barang_id	harga	jumlah
Edit Copy Delete	1	1	1	1800000	4
Edit Copy Delete	2	1	1	1800000	5
Edit Copy Delete	3	1	3	4500000	2
Edit Copy Delete	4	2	8	250000	5
Edit Copy Delete	5	2	3	4500000	4
Edit Copy Delete	6	2	2	250000	5
Edit Copy Delete	7	3	2	250000	2
Edit Copy Delete	8	3	4	350000	5
Edit Copy Delete	9	3	9	1000000	2
Edit Copy Delete	10	4	4	350000	1
Edit Copy Delete	11	4	6	75000	5
Edit Copy Delete	12	4	5	600000	1
Edit Copy Delete	13	5	8	250000	1
Edit Copy Delete	14	5	4	350000	2
Edit Copy Delete	15	5	1	1800000	5
Edit Copy Delete	16	6	6	75000	4
Edit Copy Delete	17	6	7	150000	3
Edit Copy Delete	18	6	3	4500000	5
Edit Copy Delete	19	7	10	650000	1
Edit Copy Delete	20	7	9	1000000	3
Edit Copy Delete	21	7	6	75000	1
Edit Copy Delete	22	8	2	250000	5
Edit Copy Delete	23	8	8	250000	5
Edit Copy Delete	24	8	3	4500000	2

Console

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

D. 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

a. `DB::select()`

Method ini digunakan untuk mengambil data dari database. Method ini **mengembalikan *(return)*** data hasil *query*. Contoh

```
DB::select('select * from m_user'); //Query semua data pada tabel m_user
```

```
DB::select('select * from m_user where level_id = ?', [1]); //Query tabel m_user dengan level_id = 1
```

```
DB::select('select * from m_user where level_id = ? and username = ?', [1, 'admin']);
```

b. `DB::insert()`

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

```
DB::insert('insert into m_level(level_kode, level_nama) values(?,?)', ['CUS', 'Pelanggan']);
```

c. `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

```
DB::update('update m_level set level_nama = ? where level_kode = ?', ['Customer', 'CUS']);
```

d. `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

```
DB::delete('delete from m_level where level_kode = ?', ['CUS']);
```

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`

```
php artisan make:controller LevelController
```

2. Kita modifikasi dulu untuk *routing*-nya, ada di `PWL_POS/routes/web.php`

```
<?php

use App\Http\Controllers\LevelController;
use Illuminate\Support\Facades\Route;


Route::get('/', function () {
    return view('welcome');
});

Route::get('/level', [LevelController::class, 'index']);
```

```
Route::get('/', function () {
    return view('welcome');
});

Route::get('/level', [LevelController::class,'index']);
```

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

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class LevelController extends Controller
{
    public function index()
    {
        DB::insert('insert into m_level(level_kode, level_nama, created_at) values(?, ?, ?)', ['CUS', 'Pelanggan', now()]);

        return 'Insert data baru berhasil';
    }
}
```

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class LevelController extends Controller
{
    public function index(){
        DB::insert('insert into m_level(level_kode, level_nama, created_at) values(?,?,?)', ['CUS', 'Pelanggan', now()]);
        return 'Insert data baru berhasil';
    }
}
```

4. 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`

	level_id	level_kode	level_nama	created_at	updated_at
Edit Copy Delete	1	ADM	Administrator	NULL	NULL
Edit Copy Delete	2	MNG	Manager	NULL	NULL
Edit Copy Delete	3	STF	Staff/Kasir	NULL	NULL
Edit Copy Delete	4	CUS	Pelanggan	2024-02-26 08:20:00	NULL

	level_id	level_kode	level_nama	created_at	updated_at
Edit Copy Delete	1	ADM	Administrator	NULL	NULL
Edit Copy Delete	2	MNG	Manager	NULL	NULL
Edit Copy Delete	3	STF	Staff/Kasir	NULL	NULL
Edit Copy Delete	4	CUS	Pelanggan	2025-03-01 05:26:37	NULL

5. Selanjutnya, kita modifikasi lagi file `LevelController` untuk meng-*update* data di table `m_level` seperti berikut

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class LevelController extends Controller
{
    public function index()
    {
        // DB::insert('insert into m_level(level_kode, level_nama, created_at) values(?, ?, ?)', ['CUS', 'Pelanggan', now()]);
        // return 'Insert data baru berhasil';

        $row = DB::update('update m_level set level_nama = ? where level_kode = ?', ['Customer', 'CUS']);
        return 'Update data berhasil. Jumlah data yang diupdate: ' . $row.' baris';
    }
}
```

6. 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`

```php
class LevelController extends Controller
{
    public function index()
    {
        // DB::insert('insert into m_level(level_kode, level_nama, created_at) values(?,?,?)', ['CUS', 'Pelanggan', now()]);
        // return 'Insert data baru berhasil';

        $row = DB::update('update m_level set level_nama = ? where level_kode = ?', ['Customer', 'CUS']);
        return 'Update data berhasil. Jumlah data yang diupdate'. $row .' Baris';
    }
}
```

	level_id	level_kode	level_nama	created_at	updated_at
Edit Copy Delete	1	ADM	Administrator	NULL	NULL
Edit Copy Delete	2	MNG	Manager	NULL	NULL
Edit Copy Delete	3	STF	Staff/Kasir	NULL	NULL
Edit Copy Delete	4	CUS	Customer	2025-03-01 05:26:37	NULL

Melakukan update level_nama yang sebelumnya Pelanggan menjadi Customer Menggunakan fungsi update

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

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class LevelController extends Controller
{
    public function index()
    {
        // DB::insert('insert into m_level(level_kode, level_nama, created_at) values(?, ?, ?)', ['CUS', 'Pelanggan', now()]);
        // return 'Insert data baru berhasil';

        // $row = DB::update('update m_level set level_nama = ? where level_kode = ?', ['Customer', 'CUS']);
        // return 'Update data berhasil. Jumlah data yang diupdate: ' . $row.' baris';

        $row = DB::delete('delete from m_level where level_kode = ?', ['CUS']);
        return 'Delete data berhasil. Jumlah data yang dihapus: ' . $row.' baris';
    }
}
```

```php
class LevelController extends Controller
{
    public function index(){
        // DB::insert('insert into m_level(level_kode, level_nama, created_at) values(?,?,?)', ['CUS', 'Pelanggan', now()]);
        // return 'Insert data baru berhasil';

        // $row = DB::update('update m_level set level_nama = ? where level_kode = ?', ['Customer', 'CUS']);
        // return 'Update data berhasil. Jumlah data yang diupdate'. $row .' Baris';

        $row = DB::delete('delete from m_level where level_kode = ?',['CUS']);
        return 'Delete data berhasil. Jumlah data yang dihapus '.$row.' Baris';
    }
}
```

	level_id	level_kode	level_nama	created_at	updated_at
Edit Copy Delete	1	ADM	Administrator	NULL	NULL
Edit Copy Delete	2	MNG	Manager	NULL	NULL
Edit Copy Delete	3	STF	Staff/Kasir	NULL	NULL

Menghapus baris dari tabel m_level yang dimana level_kode = CUS

8. Method terakhir yang kita coba adalah untuk menampilkan data yang ada di table `m_level`. Kita modifikasi file `LevelController` seperti berikut

```php
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class LevelController extends Controller
{
    public function index()
    {
        // DB::insert('insert into m_level(level_kode, level_nama, created_at) values(?, ?, ?)', ['CUS', 'Pelanggan', now()]);
        // return 'Insert data baru berhasil';

        // $row = DB::update('update m_level set level_nama = ? where level_kode = ?', ['Customer', 'CUS']);
        // return 'Update data berhasil. Jumlah data yang diupdate: ' . $row.' baris';

        // $row = DB::delete('delete from m_level where level_kode = ?', ['CUS']);
        // return 'Delete data berhasil. Jumlah data yang dihapus: ' . $row.' baris';

        $data = DB::select('select * from m_level');
        return view('level', ['data' => $data]);
    }
}
```

9. 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`

```
LevelController.php   level.blade.php ×   web.php
resources > views > level.blade.php > ...
<!DOCTYPE html>
<html>
    <head>
        <title>Data Level Pengguna</title>
    </head>
    <body>
        <h1>Data Level Pengguna</h1>
        <table border="1" cellpadding="2" cellspacing="0">
            <tr>
                <th>ID</th>
                <th>Kode Level</th>
                <th>Nama Level</th>
            </tr>
            @foreach ($data as $d)
            <tr>
                <td>{{ $d->level_id }}</td>
                <td>{{ $d->level_kode }}</td>
                <td>{{ $d->level_nama }}</td>
            </tr>
            @endforeach
        </table>
    </body>
</html>
```

10. Silahkan dicoba pada browser dan amati apa yang terjadi

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class LevelController extends Controller
{
    public function index(){
        // DB::insert('insert into m_level(level_kode, level_nama, created_at) values(?,?
        // return 'Insert data baru berhasil';

        // $row = DB::update('update m_level set level_nama = ? where level_kode = ?', ['
        // return 'Update data berhasil. Jumlah data yang diupdate'. $row .' Baris';

        // $row = DB::delete('delete from m_level where level_kode = ?',['CUS']);
        // return 'Delete data berhasil. Jumlah data yang dihapus '.$row.' Baris';

        $data = DB::select('select * from m_level');
        return view('level', ['data' => $data]);
    }
}

```

resources > views > level.blade.php > html > body > table

```blade
<!DOCTYPE html>
<html>
<head>
 <title>Data level Pengguna</title>
</head>
<body>
    <h1>Data Level Pengguna</h1>
    <table border="1" cellpadding="2" cellspacing="0
    <tr>
        <th>ID</th>
        <th>Kode Level</th>
        <th>Nama Level</th>
    </tr>

    @foreach ($data as $d)
        <tr>
            <td>{{$d->level_id}}</td>
            <td>{{$d->level_kode}}</td>
            <td>{{$d->level_nama}}</td>
        </tr>
    @endforeach
</table>
</body>
</html>
```

Data Level Pengguna

ID	Kode Level	Nama Level
1	ADM	Administrator
2	MNG	Manager
3	STF	Staff/Kasir

Membuat View level karena Controller memanggil level sebagai view lalu view menampilkan data yang ada pada tabel m_level dari database.

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

E. 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.

```
DB::table('<nama-tabel>'); // query builder untuk melakukan operasi CRUD pada tabel yang dituju
```

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

a. Perintah untuk *insert* data dengan method `insert()`

```
DB::table('m_kategori')->insert(['kategori_kode' => 'SMP', 'kategori_nama' => 'Smartphone']);
```

Query yang dihasilkan dari kode di atas adalah

```
insert into m_kategori(kategori_kode, kategori_nama) values('SMP', 'Smartphone');
```

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

```
DB::table('m_kategori')->where('kategori_id', 1)->update(['kategori_nama' => 'Makanan Ringan']);
```

Query yang dihasilkan dari kode di atas adalah

```
update m_kategori set kategori_nama = 'Makanan Ringan' where kategori_id = 1;
```

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

```
DB::table('m_kategori')->where('kategori_id', 9) ->delete();
```

Query yang dihasilkan dari kode di atas adalah

```
delete from m_kategori where kategori_id = 9;
```

d. 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;`

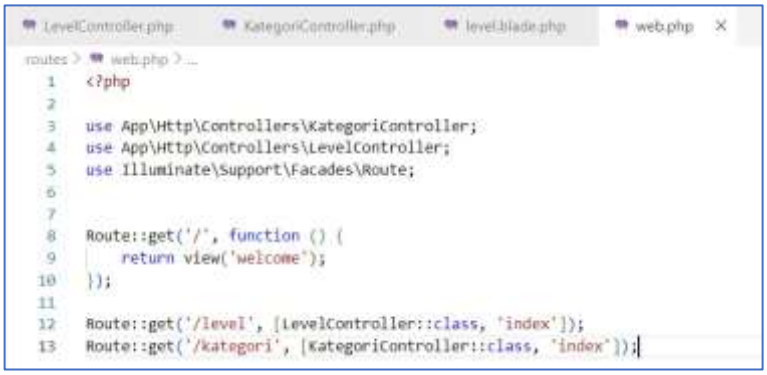
`DB::table('m_kategori')`
`->select('kategori_kode')`
`->where('kategori_id', 1)->get();`
	`select kategori_kode from m_kategori`
	`where kategori_id = 1;`

Praktikum 5 – Implementasi *Query Builder*

1. Kita buat controller dahuku untuk mengelola data pada table `m_kategori`

```
php artisan make:controller KategoriController
```

2. Kita modifikasi dulu untuk routing-nya, ada di `PWL_POS/routes/web.php`



3. Selanjutnya, kita modifikasi file `KategoriController` untuk menambahkan 1 data ke table `m_kategori`

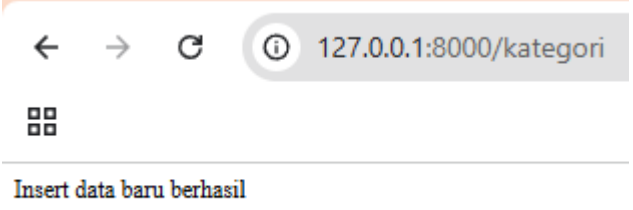
```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class KategoriController extends Controller
{
    public function index()
    {
        $data = [
            'kategori_kode' => 'SNK',
            'kategori_nama' => 'Snack/Makanan Ringan',
            'created_at' => now()
        ];
        DB::table('m_kategori')->insert($data);
        return 'Insert data baru berhasil';
    }
}
```

4. 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`

app > Http > Controllers > KategoriController.php > ...

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class KategoriController extends Controller
{
    public function index(){
        $data =[
            'kategori_kode' => 'SNK',
            'kategori_nama' => 'Snake/Makanan Ringan',
            'created_at' => now()
        ];

        DB::table('m_kategori')->insert($data);
        return 'Insert data baru berhasil';
    }
}
```

```php
Route::get('/kategori', [KategoriController::class,'index']);
```



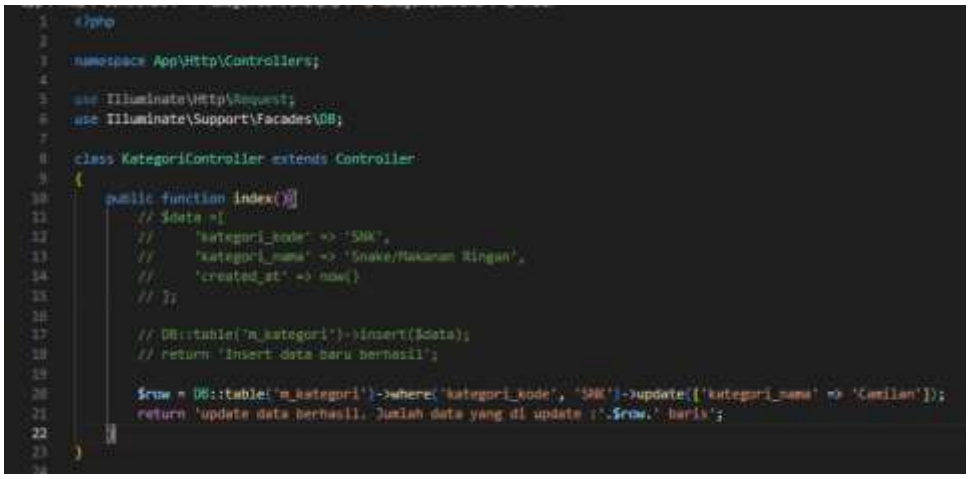
	kategori_id	kategori_kode	kategori_nama	created_at	updated_at
Edit Copy Delete	1	SPORT	Peralatan Olahraga	NULL	NULL
Edit Copy Delete	2	ELEC	Elektronik	NULL	NULL
Edit Copy Delete	3	TOOL	Perkakas	NULL	NULL
Edit Copy Delete	4	CLOTH	Pakaian	NULL	NULL
Edit Copy Delete	5	FURN	Furniture	NULL	NULL
Edit Copy Delete	6	SNK	Snake/Makanan Ringan	2025-03-01 06:07:15	NULL

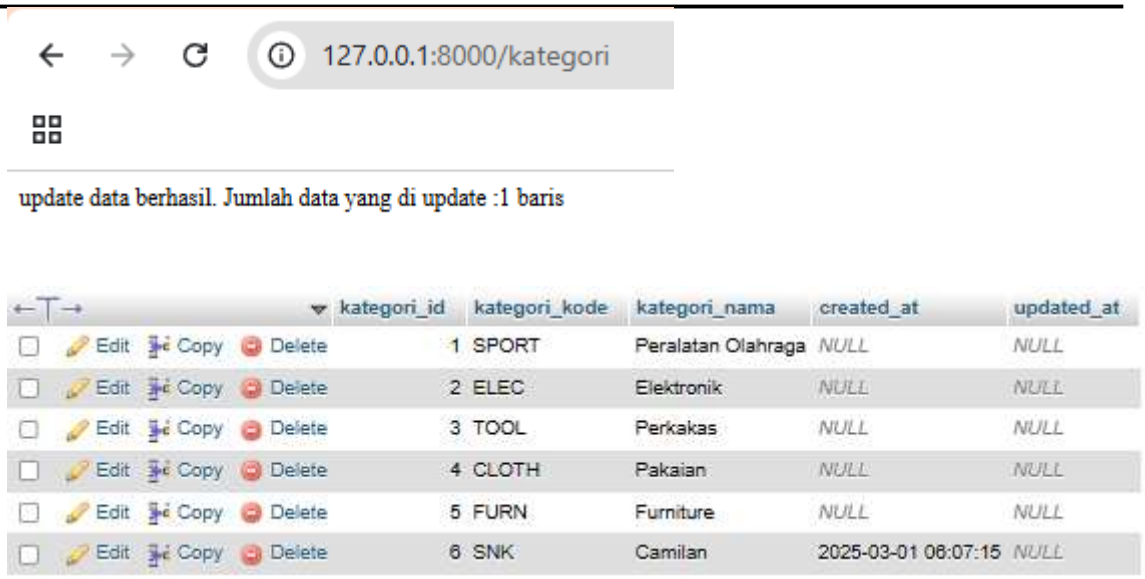
5. Selanjutnya, kita modifikasi lagi file `KategoriController` untuk meng-*update* data di table `m_kategori` seperti berikut

app > Http > Controllers > KategoriController.php > KategoriController > index

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

class KategoriController extends Controller
{
    public function index()
    {
        /* $data = [
            'kategori_kode' => 'SNK',
            'kategori_nama' => 'Snack/Makanan Ringan',
            'created_at' => now()
        ];
        DB::table('m_kategori')->insert($data);
        return 'Insert data baru berhasil'; */

        $row = DB::table('m_kategori')->where('kategori_kode', 'SNK')->update(['kategori_nama' => 'Camilan']);
        return 'Update data berhasil. Jumlah data yang diupdate: ' . $row.' baris';
    }
}
```

6. 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`



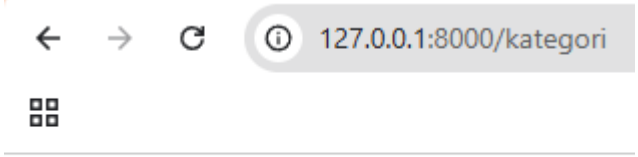


	kategori_id	kategori_kode	kategori_nama	created_at	updated_at
Edit Copy Delete	1	SPORT	Peralatan Olahraga	NULL	NULL
Edit Copy Delete	2	ELEC	Elektronik	NULL	NULL
Edit Copy Delete	3	TOOL	Perkakas	NULL	NULL
Edit Copy Delete	4	CLOTH	Pakaian	NULL	NULL
Edit Copy Delete	5	FURN	Furniture	NULL	NULL
Edit Copy Delete	6	SNK	Camilan	2025-03-01 06:07:15	NULL

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

```php
public function index()
{
    /* $data = [
        'kategori_kode' => 'SNK',
        'kategori_nama' => 'Snack/Makanan Ringan',
        'created_at' => now()
    ];
    DB::table('m_kategori')->insert($data);
    return 'Insert data baru berhasil'; */

    // $row = DB::table('m_kategori')->where('kategori_kode', 'SNK')->update(['kategori_nama' => 'Camilan']);
    // return 'Update data berhasil. Jumlah data yang diupdate: ' . $row.' baris';

    $row = DB::table('m_kategori')->where('kategori_kode', 'SNK')->delete();
    return 'Delete data berhasil. Jumlah data yang dihapus: ' . $row.' baris';
}
```

```php
    $row = DB::table('m_kategori')->where('kategori_kode', 'SNK')->delete();
    return 'Delete berhasil. Jumlah data yang di hapus '.$row.' Baris';
```



Delete berhasil. Jumlah data yang di hapus 1 Baris

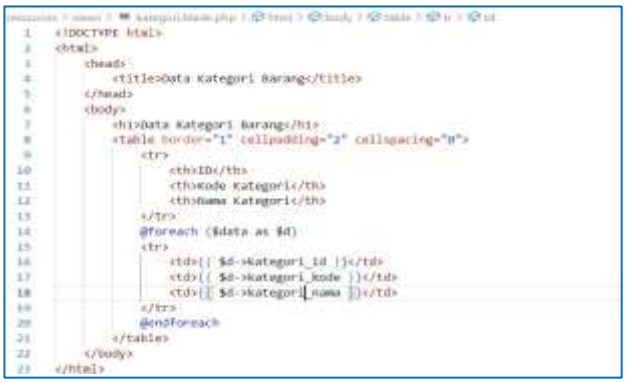
	kategori_id	kategori_kode	kategori_nama	created_at	updated_at
Edit Copy Delete	1	SPORT	Peralatan Olahraga	NULL	NULL
Edit Copy Delete	2	ELEC	Elektronik	NULL	NULL
Edit Copy Delete	3	TOOL	Perkakas	NULL	NULL
Edit Copy Delete	4	CLOTH	Pakaian	NULL	NULL
Edit Copy Delete	5	FURN	Furniture	NULL	NULL

8. Method terakhir yang kita coba adalah untuk menampilkan data yang ada di table `m_kategori`. Kita modifikasi file `KategoriController` seperti berikut

```php
public function index()
{
    /* $data = [
        'kategori_kode' => 'SNK',
        'kategori_nama' => 'Snack/Makanan Ringan',
        'created_at' => now()
    ];
    DB::table('m_kategori')->insert($data);
    return 'Insert data baru berhasil'; */

    // $row = DB::table('m_kategori')->where('kategori_kode', 'SNK')->update(['kategori_nama' => 'Camilan']);
    // return 'Update data berhasil. Jumlah data yang diupdate: ' . $row.' baris';

    // $row = DB::table('m_kategori')->where('kategori_kode', 'SNK')->delete();
    // return 'Delete data berhasil. Jumlah data yang dihapus: ' . $row.' baris';

    $data = DB::table('m_kategori')->get();
    return view('kategori', ['data' => $data]);
}
```

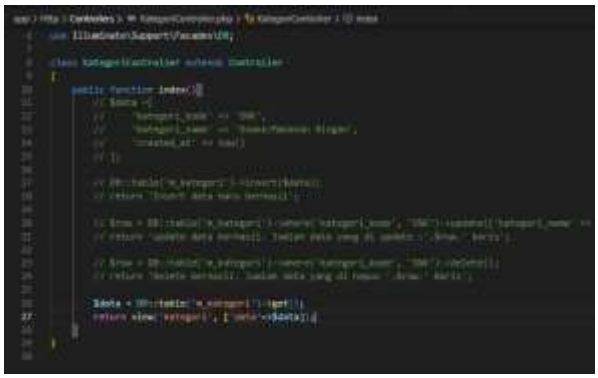
9. Coba kita perhatikan kode yang diberi tanda kotak merah, berhubung kode tersebut memanggil `view('kategori')`, maka kita buat file view pada VSCode di

PWL_POS/resources/view/kategori.blade.php

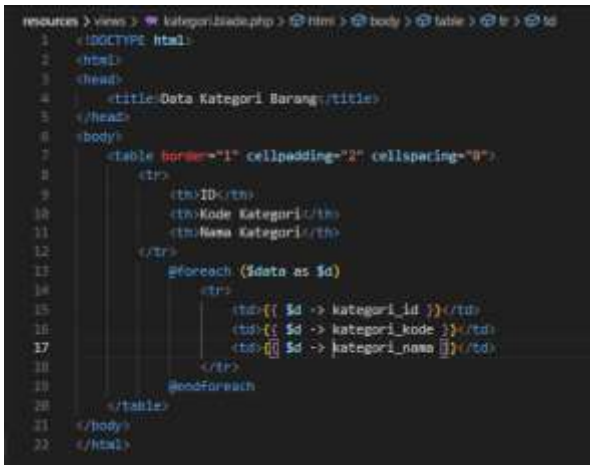


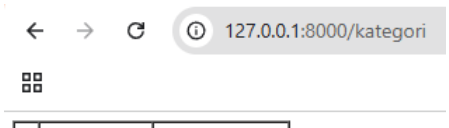
10. Silahkan dicoba pada browser dan amati apa yang terjadi.

KategoriController



View kategori





ID	Kode Kategori	Nama Kategori
1	SPORT	Peralatan Olahraga
2	ELEC	Elektronik
3	TOOL	Perkakas
4	CLOTH	Pakaian
5	FURN	Furniture

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

F. 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

```
php artisan make:model <nama-model-CamelCase>
```

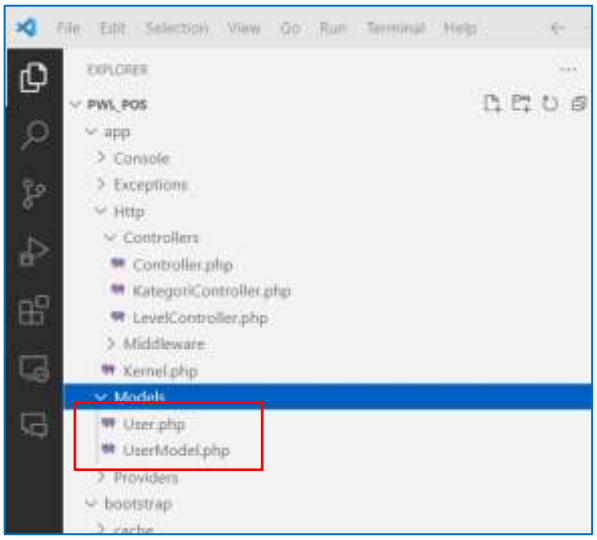
Untuk bisa melakukan operasi ***CRUD*** (*create, read/retrieve, update, delete*), kita harus membuat sebuah model sesuai dengan target tabel yang ingin digunakan. Jadi,

dalam 1 model, merepresentasikan 1 tabel database.

Praktikum 6 – Implementasi Eloquent ORM

1. Kita buat file model untuk tabel `m_user` dengan mengetikkan perintah

```
php artisan make:model UserModel
```



2. 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`
3. Kita buka file `UserModel.php` dan modifikasi seperti berikut

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class UserModel extends Model
{
    use HasFactory;

    protected $table = 'm_user';        // Mendefinisikan nama tabel yang digunakan oleh model ini
    protected $primaryKey = 'user_id';  // Mendefinisikan primary key dari tabel yang digunakan
}
```

4. Kita modifikasi route `web.php` untuk mencoba routing ke controller `UserController`

```php
<?php

use App\Http\Controllers\KategoriController;
use App\Http\Controllers\LevelController;
use App\Http\Controllers\UserController;
use Illuminate\Support\Facades\Route;


Route::get('/', function () {
    return view('welcome');
});

Route::get('/level', [LevelController::class, 'index']);
Route::get('/kategori', [KategoriController::class, 'index']);
Route::get('/user', [UserController::class, 'index']);
```

5. Sekarang, kita buat file controller `UserController` dan memodifikasinya seperti berikut

```php
<?php

namespace App\Http\Controllers;

use App\Models\UserModel;
use Illuminate\Http\Request;

class UserController extends Controller
{
    public function index()
    {
        // coba akses model UserModel
        $user = UserModel::all(); // ambil semua data dari tabel m_user
        return view('user', ['data' => $user]);
    }
}
```

6. Kemudian kita buat view `user.blade.php`

```blade.php
<!DOCTYPE html>
<html>
    <head>
        <title>Data User</title>
    </head>
    <body>
        <h1>Data User</h1>
        <table border="1" cellpadding="2" cellspacing="0">
            <tr>
                <th>ID</th>
                <th>Username</th>
                <th>Nama</th>
                <th>ID Level Pengguna</th>
            </tr>
            @foreach ($data as $d)
            <tr>
                <td>{{ $d->user_id }}</td>
                <td>{{ $d->username }}</td>
                <td>{{ $d->nama }}</td>
                <td>{{ $d->level_id }}</td>
            </tr>
            @endforeach
        </table>
    </body>
</html>
```

7. Jalankan di browser, catat dan laporkan apa yang terjadi

UserModel

app > Models > UserModel.php > UserModel

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class UserModel extends Model
{
    use HasFactory;

    protected $table = 'm_user';
    protected $primaryKey = 'user_id'; // mendefinisi primary key dari tabel

}
```

UserController

app > Http > Controllers > UserController.php > UserController > index

```php
<?php

namespace App\Http\Controllers;

use App\Models\UserModel;
use Illuminate\Http\Request;

class UserController extends Controller
{
    public function index(){
        $user = UserModel::all();
        return view('user', ['data'=>$user]);
    }
}
```

View user

resources > views > user.blade.php > html > body > table

```
<!DOCTYPE html>
<html>
<head>
    <title>Data User</title>
</head>
<body>
    <h1>Data User</h1>
    <table border="1" cellpadding="2">
        <tr>
            <th>ID</th>
            <th>Username</th>
            <th>Nama</th>
            <th>IDPengguna</th>
        </tr>

        @foreach ($data as $d)
            <tr>
                <td>{{$d->user_id}}</td>
                <td>{{$d->username}}</td>
                <td>{{$d->nama}}</td>
                <td>{{$d->level_id}}</td>
            </tr>
        @endforeach
    </table>
</body>
</html>
```

8. Setelah itu, kita modifikasi lagi file `UserController`

app > Http > Controllers > UserController.php > ...

```
<?php

namespace App\Http\Controllers;

use App\Models\UserModel;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;

class UserController extends Controller
{
    public function index()
    {
        // tambah data user dengan Eloquent Model
        $data = [
            'username' => 'customer-1',
            'nama' => 'Pelanggan',
            'password' => Hash::make('12345'),
            'level_id' => 4
        ];
        UserModel::insert($data); // tambahkan data ke tabel m_user

        // coba akses model UserModel
        $user = UserModel::all(); // ambil semua data dari tabel m_user
        return view('user', ['data' => $user]);
    }
}
```

9. Jalankan di browser, amati dan laporkan apa yang terjadi

```php
class UserController extends Controller
{

    public function index(){

        $data = [
            'username' => 'customer-1',
            'nama' => 'Pelanggan',
            'password' => Hash::make('12345'),
            'level_id' => 6
        ];

        UserModel::insert($data);

        $user = UserModel::all();
        return view('user', ['data'=>$user]);
    }
}
```

Data User

ID	Username	Nama	IDPengguna
1	admin	Administrator	1
2	manager	Manager	2
3	staff	Staff/Kasir	3
11	customer-1	Pelanggan	6

10. Kita modifikasi lagi file `UserController` menjadi seperti berikut

```php
class UserController extends Controller
{
    public function index()
    {
        // tambah data user dengan Eloquent Model
        $data = [
            'nama' => 'Pelanggan Pertama',
        ];
        UserModel::where('username', 'customer-1')->update($data); // update data user

        // coba akses model UserModel
        $user = UserModel::all(); // ambil semua data dari tabel m_user
        return view('user', ['data' => $user]);
    }
}
```

11. Jalankan di browser, amati dan laporkan apa yang terjadi

app > Http > Controllers > UserController.php > UserController > index

```php
namespace App\Http\Controllers;

use App\Models\UserModel;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;

class UserController extends Controller
{

    public function index(){

        $data = [
            'nama' => 'Pelanggan Pertama',
        ];


        UserModel::where('username', 'customer-1')->update($data);

        $user = UserModel::all();
        return view('user', ['data'=>$user]);
    }
}
```

Data User

ID	Username	Nama	IDPengguna
1	admin	Administrator	1
2	manager	Manager	2
3	staff	Staff/Kasir	3
11	customer-1	Pelanggan Pertama	6

Mengganti nama pada tabel m_user yang bernama pelanggan menjadi Pelanggan Pertama

12. Jika sudah, laporkan hasil Praktikum-6 ini dan *commit* perubahan pada *git*

G. 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?
   - Untuk meng enkripsi data
2. Pada **Praktikum 1**, bagaimana kita men-*generate* nilai untuk `APP_KEY`?
   - Dengan command php artisan key:generate pada terminal
3. Pada **Praktikum 2.1 - Tahap 1**, secara *default* Laravel memiliki berapa file migrasi? dan untuk apa saja file migrasi tersebut?
   - 4 file migrasi

2014_10_12_000000_create_users_table.php berfungsi untuk membuat tabel user untuk menyimpan data pengguna.

2014_10_12_100000_create_password_reset_tokens_table.php untuk token reset password pengguna.

2019_08_19_000000_create_failed_jobs_table.php berfungsi untuk menyimpan informasi jobs yang gagal.

2019_12_14_000001_create_personal_access_tokens_table.php Menyimpan token API.

4. Secara *default*, file migrasi terdapat kode `$table->timestamps();`, apa tujuan/*output* dari fungsi tersebut?
   - Menunjukkan kapan data dibuat / di update.
5. Pada File Migrasi, terdapat fungsi `$table->id();` Tipe data apa yang dihasilkan dari fungsi tersebut?
   - Membuat kolom primarykey.
6. Apa bedanya hasil migrasi pada table `m_level`, antara menggunakan `$table->id();` dengan menggunakan `$table->id('level_id');` ?
   - Hanya berbeda pada penamaan kolomnya saja.
7. Pada migration, Fungsi `->unique()` digunakan untuk apa?
   - Menghindari kesamaan suatu data.
8. 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')` ?
   - Untuk meng indexing / mendeklarasikan foreignkey.
9. Pada **Praktikum 3 - Tahap 6**, apa tujuan dari Class `Hash`? dan apa maksud dari kode program `Hash::make('1234');`?
   - Hash memiliki tujuan untuk mengenkripsi suatu data tertentu seperti pada password
10. Pada **Praktikum 4 - Tahap 3/5/7**, pada *query builder* terdapat tanda tanya (`?`), apa kegunaan dari tanda tanya (`?`) tersebut?
    - (?) digunakan untuk parameter mengisi data mana saja yang akan di insert.
11. Pada **Praktikum 6 - Tahap 3**, apa tujuan penulisan kode `protected $table = 'm_user';` dan `protected $primaryKey = 'user_id';` ?
    - Agar mimiliki keamanan tapi tetap memungkinkan subclass bisa mengakses nya.
12. Menurut kalian, lebih mudah menggunakan mana dalam melakukan operasi CRUD ke database (*DB Façade / Query Builder / Eloquent ORM*) ? jelaskan
    - Menurut saya lebih mudah menggunakan Query builder,karena penulisan sintaksnya hampir sama dengan penulisan query pada database itu sendiri(secara langsung)

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