# MODUL 12.1
## MIGRATION

Tujuan Praktikum:

1. Memahami fungsi dan cara kerja model pada framework laravel
2. Mampu menerapkan migration pada aplikasi

### 12.1.1. DASAR TEORI

Migration pada Laravel merupakan sebuah fitur yang dapat membantu programmer mengelola database secara efisien dengan menggunakan kode. Migration membantu programmer dalam membuat (create), mengubah (edit), dan menghapus (delete) struktur tabel dan kolom pada database milik kita dengan cepat dan mudah.

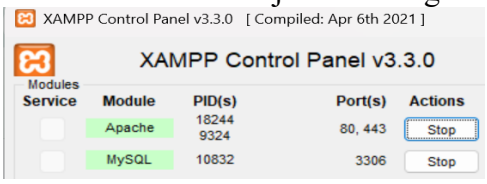
Dengan Migration kita dapat melakukan perubahan pada struktur database tanpa harus menghapus data yang ada. Migration membuat pengembangan aplikasi menjadi lebih mudah terutama dalam mengatasi perubahan kebutuhan yang muncul selama proses pengembangan aplikasi kita.

Migration memungkinkan kita untuk mengelola perubahan pada struktur database dan menyimpannya pada file yang dapat dijalankan dengan mudah di lingkungan lain. Dalam istilah yang lebih teknis, migration disebut sebagai version control untuk database. Selain dipakai membuat tabel, kita juga bisa membuat step-by-step perubahan struktur tabel.

### 12.1.2. PEMBAHASAN

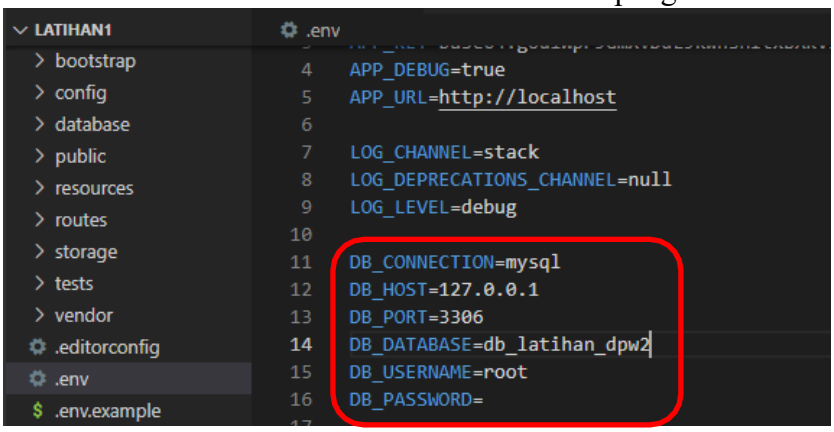
### 12.1.3.6. File Konfigurasi

Sebelum menjalankan migration dan database, jalankan Apache dan MySQL di XAMPP.

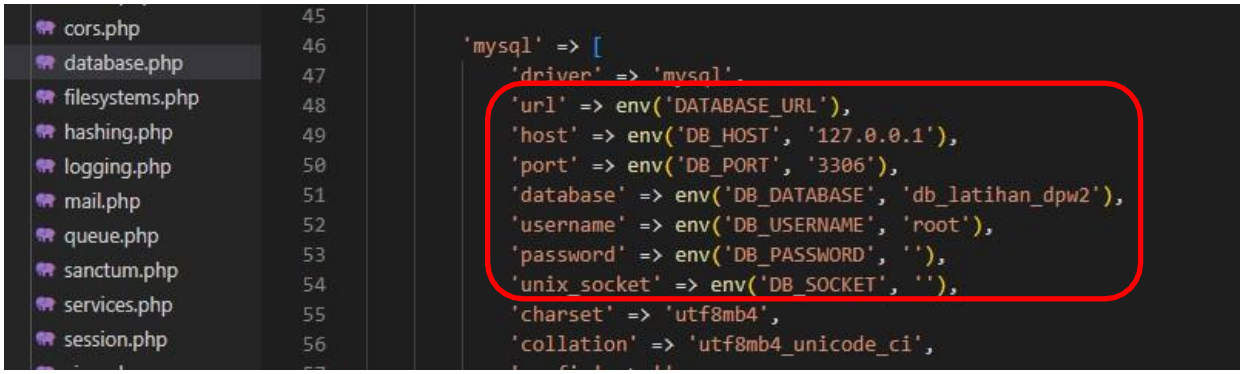



Pada laravel terdapat dua file konfigurasi yang perlu di atur: pertama konfigurasi lokal di file **.env** dan kedua pengaturan global di folder **config.**

Silahkan buka file .env kemudian lakukan pengaturan database seperti berikut:

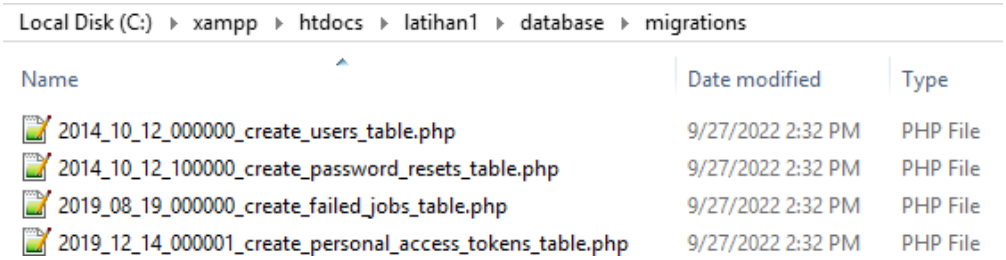



Selanjutnya lakukan konfigurasi kedua, yaitu di **config\database.php.** lakukan pengaturan database seperti berikut:




### 12.1.3.7. File migration bawaan laravel

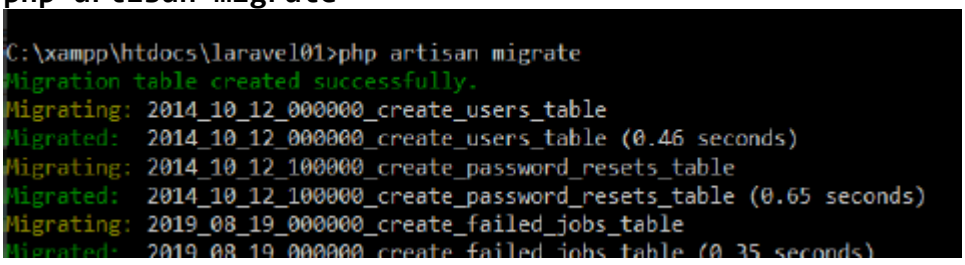
Dalam beberapa versi Laravel sudah terdapat beberapa file migration bawaan. File ini berada di folder **database\migrations**.




Setiap file migration dipakai untuk membuat 1 tabel. Artinya, 4 file migration ini akan Membuat 4 buah tabel ke dalam database.
Nama file migration diawali dengan timestamp, yakni waktu ketika file tersebut dibuat. Dalam contoh di atas, file migration 2014_10_12_000000_create_users_table.php dibuat pada tanggal 12 Oktober 2014.

Coba jalankan file migration bawaan Laravel. Keempat file migration ini akan membuat tabel users, password_resets, failed_jobs dan access_token ke dalam database laravel yang sudah kita siapkan. Untuk menjalankan migration, silahkan buka cmd, masuk ke folder laravel dan jalankan perintah berikut:

```
php artisan migrate
```




Untuk mengetahui apakah database sudah terbentuk, silahkan buka XAMPP dan nyalakan Apache dan MySQL, lalu cek database mysql menggunakan phpmyadmin.

### 12.1.3.8. Migration Rollback

Pada saat menjalankan perintah php artisan migrate, file migration akan di eksekusi secara berurutan sesuai tanggal timestamp. File dengan timestamp terbaru akan dijalankan paling akhir.

Pada prakteknya, bisa saja file migration ini ditambah secara bertahap. Misalkan programmer A sedang membuat fitur baru dan butuh tambahan sebuah tabel, maka dia bisa membuat dan menjalankan file migration untuk tabel_a. Esoknya programmer B juga bisa membuat file migration lain untuk tabel_b, dst.

Jika tiba-tiba terjadi masalah akibat penambahan tabel ini, kita bisa melakukan proses rollback sebagian terhadap file migration. Jadi seolah-olah kembali ke struktur database di hari sebelumnya.

Untuk melihat daftar urutan migration, bisa dari perintah **`php artisan migrate:status`**

Jika saya ingin mengembalikan posisi sebelum file migration terakhir, bisa menggunakan perintah **`php artisan migrate:rollback --step=1`**

Perintah ini akan me-rollback 1 file migration terakhir dan juga menghapus tabelnya.

## 12.1.3. LATIHAN

### 12.1.3.1. Membuat Migration

Sama seperti controller, file migration bisa dibuat dari teks editor. Caranya, copy kode dari file migration yang sudah ada, lalu edit di bagian yang diperlukan. Atau cara yang lebih praktis adalah dari perintah php artisan dengan format berikut:

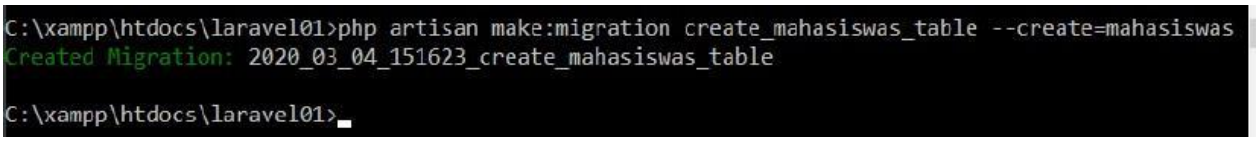
**php artisan make:migration <nama_migration> --create=<nama_tabel>**

Untuk nama_migration bisa di isi suka-suka, namun penulisan terbaik adalah dengan format:

**<nama_proses>_<nama_tabel(s)>_table**

Sebagai contoh, untuk membuat file migration tabel mahasiswas, bisa dengan menjalankan perintah berikut:

```
php artisan make:migration create_mahasiswas_table --create=mahasiswas
```




Hasilnya, di dalam folder database\migrations akan tampil file 2020_03_04_151623_create_mahasiswas_table.php

Nama file ini akan berbeda karena tanggal timestamp di generate secara otomatis yang diambil dari sistem komputer.

Akses file migration ....._**create_mahasiswas_table.php** dan ubah kode seperti berikut:

```php
<?php

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

class CreateMahasiswasTable extends Migration
{

  public function up(){
  Schema::create('mahasiswas', function (Blueprint $table) {
                        $table->id();
                        $table->char('nim',8);
                        $table->string('nama');
                        $table->string('tempat_lahir');
                        $table->date('tanggal_lahir');
                        $table->string('fakultas');
                        $table->string('jurusan');
                        $table->decimal('ipk',3,2);
                        $table->timestamps();
        });
  }
    public function down(){
    Schema::dropIfExists('mahasiswas');
  }
}
```

Jalankan kembali **`php artisan migrate.`**

Struktur tabel yang terbentuk di phpmyadmin adalah seperti berikut:

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| id | bigint(20) unsigned | NO | PRI | NULL | auto_increment |
| nim | char(8) | NO | | NULL | |
| nama | varchar(255) | NO | | NULL | |
| tempat_lahir | varchar(255) | NO | | NULL | |
| tanggal_lahir | date | NO | | NULL | |
| fakultas | varchar(255) | NO | | NULL | |
| jurusan | varchar(255) | NO | | NULL | |
| ipk | decimal(3,2) | NO | | NULL | |
| created_at | timestamp | YES | | NULL | |
| updated_at | timestamp | YES | | NULL | |

### 12.1.3.2. Modifikasi file migration

File migration yg sudah dibuat dapat diubah sesuai dengan kebutuhan, berikut contoh perubahan pada kolom nim dan ip.

```
Schema::create('mahasiswas', function (Blueprint $table) {
            $table->id();
            $table->char('nim',8)->unique();
            $table->string('nama');
            $table->string('tempat_lahir');
            $table->date('tanggal_lahir');
            $table->string('fakultas');
            $table->string('jurusan');
            $table->decimal('ipk',3,2)->default(1.00);
            $table->timestamps();
        });
    }
```

Agar perubahan ini bisa diterapkan ke tabel mahasiswas, **silahkan rollback, lalu migrate** kembali. Dan berikut hasil dari query **`DESC mahasiswas`** di SQL phpmyadmin:

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| id | bigint(20) unsigned | NO | PRI | NULL | auto_increment |
| nim | char(8) | NO | UNI | NULL | |
| nama | varchar(255) | NO | | NULL | |
| tempat_lahir | varchar(255) | NO | | NULL | |
| tanggal_lahir | date | NO | | NULL | |
| fakultas | varchar(255) | NO | | NULL | |
| jurusan | varchar(255) | NO | | NULL | |
| ipk | decimal(3,2) | NO | | 1.00 | |
| created_at | timestamp | YES | | NULL | |
| updated_at | timestamp | YES | | NULL | |

### 12.1.3.3. Alter table migration

Migration tidak hanya dipakai untuk membuat dan menghapus tabel, tapi juga untuk **modifikasi struktur tabel (menjalankan query ALTER).** Ini cukup sering kita lakukan sepanjang pembuatan project.

Misalnya tiba-tiba client memberitahu ada perubahan data yang butuh modifikasi struktur tabel. Sebagai antisipasi, kita bisa membuat file migration baru yang berisi perintah modifikasi tersebut. Jika ternyata ada masalah, dengan 1 perintah rollback, tabel bisa kembali ke bentuk semula.

Agar bisa melakukan modifikasi tabel ke dalam migration, Laravel butuh sebuah library tambahan bernama **Doctrine DBAL**. Proses instalasi library ini sangat mudah, cukup jalankan perintah berikut di folder laravel:

**composer require doctrine/dbal**

```
C:\xampp\htdocs\laravel01>composer require doctrine/dbal
Using version ^2.10 for doctrine/dbal
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 3 installs, 0 updates, 0 removals
  - Installing doctrine/event-manager (1.1.0): Downloading (100%)
  - Installing doctrine/cache (1.10.0): Downloading (100%)
  - Installing doctrine/dbal (v2.10.1): Downloading (100%)
```

Selanjutnya memodifikasi struktur tabel mahasiswas. Silahkan buat file migration baru dengan nama alter_mahasiswas_table:

**php artisan make:migration alter_mahasiswas_table --table=mahasiswas**

```
C:\xampp\htdocs\laravel01>php artisan make:migration alter_mahasiswas_table --table=mahasiswas
Created Migration: 2020_03_04_154427_alter_mahasiswas_table
```

Kemudian buka file migration ini di teks editor. Terlihat method up() dan down() tidak berisi kode apapun karena Laravel bisa mendeteksi kalau tabel mahasiswas sudah ada, sehingga tidak mungkin file migration juga dipakai pembuatan tabel lagi.

Modifikasi file **..._alter_mahasiswas_table.php** migration sebagai berikut:

```php
<?php

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

class AlterMahasiswasTable extends Migration
  {

  public function up(){
                Schema::table('mahasiswas', function (Blueprint $table) {
                $table->renameColumn('nama','nama_lengkap');
                $table->text('alamat')->after('tanggal_lahir');
                $table->dropColumn('ipk');
        });
  }

  public function down(){
                Schema::table('mahasiswas', function (Blueprint $table) {
                $table->renameColumn('nama_lengkap','nama');
                $table->dropColumn('alamat');
                $table->decimal('ipk',3,2)->default(1.00);
                });
        }
  }
```

Di dalam method up() saya membuat 3 buah perintah modifikasi:

- Method $table->renameColumn('nama','nama_lengkap') dipakai untuk mengubah nama kolom 'nama' menjadi 'nama_lengkap'.
- Method $table->text('alamat')->after('tanggal_lahir') dipakai untuk menambah kolom 'alamat' dengan tipe data TEXT, yang posisinya ditempatkan setelah kolom 'tanggal_lahir'.
- Method $table->dropColumn('ipk') dipakai untuk menghapus kolom 'ipk'.

Ketiga perubahan di method up() ini harus kita balik di method down():

- Method $table->renameColumn('nama_lengkap','nama') dipakai untuk mengubah kembali nama kolom dari 'nama_lengkap' menjadi 'nama'.
- Method $table->dropColumn('alamat') dipakai untuk menghapus kolom 'alamat'.
- Method $table->decimal('ipk',3,2)->default(1.00) dipakai untuk membuat kembali kolom 'ipk' dengan tipe data DECIMAL(3,2) dan nilai default 1.00.

Kode program di dalam method up() dan down() harus berpasangan agar proses rollback bisa berlangsung dengan baik.
Jalankan kembali migration perubahan ini dengan perintah **`php artisan migrate`**

Struktur tabel baru akan berubah menjadi seperti berikut:

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| id | bigint(20) unsigned | NO | PRI | NULL | auto_increment |
| nim | char(8) | NO | UNI | NULL | |
| nama_lengkap | varchar(255) | NO | | NULL | |
| tempat_lahir | varchar(255) | NO | | NULL | |
| tanggal_lahir | date | NO | | NULL | |
| alamat | text | NO | | NULL | |
| fakultas | varchar(255) | NO | | NULL | |
| jurusan | varchar(255) | NO | | NULL | |
| created_at | timestamp | YES | | NULL | |
| updated_at | timestamp | YES | | NULL | |

Kolom 'nama' sudah berubah jadi 'nama_lengkap', kolom 'alamat' juga telah ditambahkan dan berada setelah kolom 'tanggal_lahir', serta kolom 'ipk' juga tidak lagi ditemukan.

# MODUL 12.2
**(DB Facade , Eloquent ORM, Query Builder)**

## 12.2.1. TUJUAN

1. Memahami fungsi dan dapat mengimplementasikan model pada framework laravel
2. Dapat memahami dan mengimplementasikan fitur DB Facade (Raw SQL Queries)
3. Dapat memahami dan mengimplementasikan fitur *Eloquent ORM* (*Object-Relational Mapping*)
4. Dapat memahami dan mengimplementasikan fitur *Query Builder.*

## 12.2.2. DASAR TEORI

**Model** pada laravel merupakan representasi dari tabel database. Dengan menggunakan model, pengembang dapat melakukan operasi-operasi database seperti membuat, membaca, mengupdate, dan menghapus atau istilah lainya CRUD (*Create, Read, Update, Delete*) data dengan sangat mudah dan efisien.

**Raw query** merupakan cara paling dasar dan 'paling tradisional' di dalam Laravel, terutama jika dibandingkan dengan query builder dan eloquent ORM. Raw query juga sangat familiar karena sudah biasa kita pakai ketika membuat aplikasi PHP tanpa framework. Selain itu untuk query yang kompleks, kadang hanya bisa dijalankan dengan raw query.

**Eloquent ORM** (*Object-Relational Mapping*) yang memungkinkan pengembang untuk berinteraksi dengan database menggunakan objek dan metode-metode yang intuitif yang membuat pengembangan aplikasi menjadi lebih cepat dan efisien.

**Query Builder** digunakan untuk membuat query SQL dengan lebih fleksibel dan aman. Pengembang dapat dengan mudah membuat query kompleks menggunakan metode-metode yang disediakan oleh Query Builder.

Beberapa alasan mengapa model pada laravel dianggap penting:

1. Keamanan Data: Dengan menggunakan Model, pengembang dapat menghindari serangan *SQL Injection* dan *Cross-Site Scripting* (XSS) karena Laravel secara otomatis melindungi aplikasi dari serangan-serangan tersebut.
2. Kode yang Bersih dan Rapi: Penggunaan Model membuat kode program menjadi lebih bersih dan rapi. Model memungkinkan pengembang untuk mengorganisir logika bisnis aplikasi dengan baik, memisahkan logika database dari logika aplikasi.
3. Kemudahan Pengembangan: Model mempercepat proses pengembangan dengan menyediakan berbagai fitur yang sudah siap pakai. Pengembang tidak perlu menulis query SQL yang kompleks secara manual, ini menghemat waktu dan usaha pengembang.

## 12.2.3. PEMBAHASAN

### 12.2.3.1. DB Facade (*Raw SQL Queries*)

Raw SQL atau perintah query mentah atau disebut sebagai raw (mentah), karena query langsung ditulis sebagaimana yang biasa diinput ke dalam mysqli extension atau PDO. Query yang dimaksud adalah perintah SQL seperti

```
‘SELECT * FROM mahasiswas', 'INSERT INTO mahasiswas...', 'UPDATE mahasiswas
```

```
SET...', dst.
```

#### 12.2.3.2. Query Builder

Query builder adalah interface khusus yang disediakan Laravel untuk mengakses database. Berbeda dengan raw query dimana kita menulis langsung perintah query SQL, di dalam query builder perintah SQL ini diakses menggunakan method. Artinya, kita tidak menulis langsung perintah SQL, tapi hanya memanggil method-method saja.

#### 12.2.3.3. Eloquent ORM

Eloquent ORM adalah cara pengaksesan database dimana setiap baris tabel dianggap sebagai sebuah object. Kata ORM sendiri merupakan singkatan dari Object-relational mapping, yakni sebuah teknik programming untuk mengkonversi data ke dalam bentuk object.
Sebagaimana yang sudah kita pahami, database terdiri dari kumpulan tabel yang saling terhubung. Di dalam setiap tabel, data disimpan dalam bentuk baris dan kolom. ORM dipakai untuk mengubah baris dan kolom ini menjadi sebuah object. Nantinya, setiap kolom akan menjadi property dari object tersebut.

### 12.2.4. LATIHAN

#### 12.2.4.1 Input Data menggunakan raw SQL Queries

Sebelum masuk ke proses insert data, silahkan buat route dan controller terlebih dahulu, data yang akan digunakan diambil dari tabel mahasiswas yang telah dibuat pada pertemuan sebelumnya, jika belum ada tabel mahasiswas silahkan dibuat terlebih dahulu.

Buat controllers dengan nama **MahasiswaController**
Jalankan perintah ini di terminal: **`php artisan make:controller MahasiswaController`**

Modifikasi isinya seperti di bawah ini:

```php
<?php

namespace App\Http\Controllers;

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

class MahasiswaController extends Controller{
    public function index(){
        return "Index untuk mahasiswa";
    }

    public function insertSql() {
        $result1 = DB::insert("
        INSERT INTO mahasiswas
        (nim, nama_lengkap, tempat_lahir, tanggal_lahir, alamat, fakultas,
jurusan)
```

```
        VALUES
        ('20104065',
         'Muhammad Nur Hamada',
         'Bandung',
         '2002-02-02',
         'Jl. Contoh No. 123',
         'Fakultas Informatika',
         'Software Engineering')"
        );
        dump($result1);
    }
}
```

**dump** = menampilkan hasil eksekusi $result1

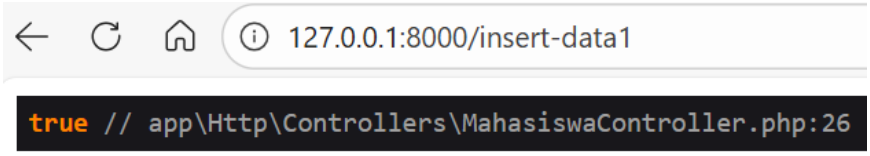
Buat root untuk insert data di routes\web.php seperti berikut:

```
<?php

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

Route::get('/insert-data1', [MahasiswaController::class,'insertSql']);
```

Untuk menjalankan perintah silahkan akses akses alamat **localhost:8000/insert-data1**, jika berhasil maka laravel akan menampilkan hasil sebagai berikut:




Jika true, data berarti telah tersimpan di database tabel mahasiswas.

Laravel menyediakan facade DB untuk mengeksekusi perintah SQL mentah seperti **INSERT, UPDATE, DELETE, dan SELECT.**

| Operasi | Command | Return |
|---|---|---|
| Insert | DB::insert() | true/false |
| Update | DB::update() | jumlah baris |
| Delete | DB::delete() | jumlah baris |
| Select | DB::select() | array of object |

**Insert:**

```
DB::insert("
    INSERT INTO nama_tabel (kolom1, kolom2, kolom3)
    VALUES ('nilai1', 'nilai2', 'nilai3')
");
```

**Update:**

```
DB::update("
    UPDATE nama_tabel
    SET kolom1 = 'nilai1',
        kolom2 = 'nilai2'
    WHERE kondisi
");
```

**Delete:**

```
DB::delete("
    DELETE FROM nama_tabel
    WHERE kondisi
");
```

**Select:**

```
DB::select("
    SELECT kolom1, kolom2
    FROM nama_tabel
    WHERE kondisi
");
```

### 12.2.4.2 Input Data menggunakan Query Builder

Selanjutnya input data menggunakan query builder, gunakan route dan controllers yang sama untuk membuat query builder. Ganti baris kode untuk insert data pada raw queries menjadi seperti berikut:

Tambahkan fungsi ini di MahasiswaController.php:

```
public function insertQB()
    {
        $result2 = DB::table('mahasiswas')->insert([
            'nim'           => '20104070',
            'nama_lengkap'  => 'Aulia Putri Ramadhani',
            'tempat_lahir'  => 'Surabaya',
            'tanggal_lahir' => '2003-05-14',
            'alamat'        => 'Jl. Mawar No. 10',
            'fakultas'      => 'Fakultas Teknik',
            'jurusan'       => 'Teknik Informatika',
            ]);
        dump($result2);
    }
```

Tambahkan route baru di routes\web.php:

```
Route::get('/insert-data2', [MahasiswaController::class,'insertQB']);
```

Untuk menjalankan perintah silahkan akses akses alamat **localhost:8000/insert-data2**, jika berhasil maka data baru akan tersimpan di database tabel mahasiswas.

Berikut ini perintah lainnya untuk eksekusi Query Buildier:

**INSERT – Query Builder:**

```
DB::table('nama_tabel')->insert([
    'kolom1' => 'nilai1',
    'kolom2' => 'nilai2',
    'kolom3' => 'nilai3',
]);
```

**UPDATE – Query Builder:**

```
DB::table('nama_tabel')
    ->where('kondisi_kolom', 'nilai')
    ->update([
        'kolom1' => 'nilai1',
        'kolom2' => 'nilai2',
    ]);
```

**DELETE – Query Builder**

```
DB::table('nama_tabel')
    ->where('kondisi_kolom', 'nilai')
    ->delete();
```

**SELECT – Query Builder**

Ambil semua data:

```
DB::table('nama_tabel')->get();
```

Dengan kondisi:

```
DB::table('nama_tabel')
    ->where('kondisi_kolom', 'nilai')
   ->get();
```

### 12.2.4.3 Input Data menggunakan Eloquent ORM

Selanjutnya input data menggunakan Eloquent ORM, gunakan route dan controllers yang sama untuk membuat Eloquent ORM.

Buat Model dengan Nama **Mahasiswa.php** dengan command:
**`php artisan make:model Mahasiswa`**

Modifikasi isi **\app\Models\Mahasiswa.php** seperti berikut:

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Mahasiswa extends Model
{
    protected $table = 'mahasiswas';

    protected $fillable = [
        'nim',
        'nama_lengkap',
        'tempat_lahir',
        'tanggal_lahir',
        'alamat',
        'fakultas',
        'jurusan',
    ];
}
```

**Eloquent ORM** menggunakan **model** sebagai **representasi tabel database**.
Properti **$table** digunakan untuk **menentukan nama tabel** yang direpresentasikan, sedangkan **$fillable** berfungsi untuk **membatasi kolom yang dapat diisi** melalui mekanisme mass assignment guna meningkatkan keamanan data.

Buat function baru di **MahasiswaController.php:**

```php
use App\Models\Mahasiswa;

public function insertEloquent()
    {
        $mhs = Mahasiswa::create([
            'nim'           => '20104080',
            'nama_lengkap'  => 'Rizky Ananda',
            'tempat_lahir'  => 'Malang',
            'tanggal_lahir' => '2002-09-12',
            'alamat'        => 'Jl. Kenanga No. 7',
            'fakultas'      => 'Fakultas Informatika',
            'jurusan'       => 'Software Engineering',
        ]);
        dump($mhs);
    }
```

Untuk menjalankan perintah silahkan akses akses alamat **localhost:8000/insert-data**, jika berhasil maka data baru akan tersimpan di database tabel mahasiswas.

Perintah lain untuk Eloquent ORM:

**INSERT – Eloquent**

```php
Model::create([
        'kolom1' => 'nilai1',
        'kolom2' => 'nilai2',
]);
```

**SELECT – semua data**

```php
Model::all();
```

**SELECT – dengan kondisi**

```php
Model::where('kondisi_kolom', 'nilai')->get();
```

**UPDATE – Eloquent**

```php
Model::where('kondisi_kolom', 'nilai')
        ->update([
            'kolom1' => 'nilai1',
            'kolom2' => 'nilai2',
        ]);
```

**DELETE – Eloquent**

```php
Model::where('kondisi_kolom', 'nilai')->delete();
```

# TUGAS

## 12.1

Buatlah file migration untuk database yang akan digunakan pada tugas besar di kelas teori.

## 12.2

Buatlah fungsi-fungsi berikut untuk menyelesaikan tugas besar di matakuliah teori:

1. Buat fungsi untuk insert data menggunakan **raw SQL Queries**
2. Buat fungsi untuk insert data menggunakan **Query Builder**
3. Buat fungsi untuk insert data menggunakan **Eloquent ORM**