MIGRATION

Mapel MKK RPL Semester Genap

Migrations are like version control for your database, allowing your team to define and share the application's database schema definition. If you have ever had to tell a teammate to manually add a column to their local database schema after pulling in your changes from source control, you've faced the problem that database migrations solve.

```
php artisan make:migration create_flights_table
```

To run all of your outstanding migrations, execute the `migrate` Artisan command:

```
php artisan migrate
```

If you would like to see which migrations have run thus far, you may use the `migrate:status` Artisan command:

```
php artisan migrate:status
```

```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('flights', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('airline');
            $table->timestamps();
        });
    }

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

Creating Tables

To create a new database table, use the `create` method on the `Schema` facade. The `create` method accepts two arguments: the first is the name of the table, while the second is a closure which receives a `Blueprint` object that may be used to define the new table:

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

Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email');
    $table->timestamps();
});
```

Renaming / Dropping Tables

To rename an existing database table, use the `rename` method:

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

Schema::rename($from, $to);
```

To drop an existing table, you may use the `drop` or `dropIfExists` methods:

```php
Schema::drop('users');

Schema::dropIfExists('users');
```

Creating Columns

The `table` method on the `Schema` facade may be used to update existing tables. Like the `create` method, the `table` method accepts two arguments: the name of the table and a closure that receives an `Illuminate\Database\Schema\Blueprint` instance you may use to add columns to the table:

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

Schema::table('users', function (Blueprint $table) {
    $table->integer('votes');
});
```

Renaming Columns

To rename a column, you may use the `renameColumn` method provided by the schema builder:

```
Schema::table('users', function (Blueprint $table) {
    $table->renameColumn('from', 'to');
});
```

Dropping Columns

To drop a column, you may use the `dropColumn` method on the schema builder:

```
Schema::table('users', function (Blueprint $table) {
    $table->dropColumn('votes');
});
```

You may drop multiple columns from a table by passing an array of column names to the `dropColumn` method:

```
Schema::table('users', function (Blueprint $table) {
    $table->dropColumn(['votes', 'avatar', 'location']);
});
```

Available Column Types

The schema builder blueprint offers a variety of methods that correspond to the different types of columns you can add to your database tables. Each of the available methods are listed in the table below:

bigIncrements	jsonb	string
bigInteger	lineString	text
binary	longText	timeTz
boolean	macAddress	time
char	mediumIncrements	timestampTz
dateTimeTz	mediumInteger	timestamp
dateTime	mediumText	timestampsTz
date	morphs	timestamps
decimal	multiLineString	tinyIncrements
double	multiPoint	tinyInteger
enum	multiPolygon	tinyText
float	nullableMorphs	unsignedBigInteger
foreignId	nullableTimestamps	unsignedDecimal
foreignIdFor	nullableUlidMorphs	unsignedInteger
foreignUlid	nullableUuidMorphs	unsignedMediumInteger
foreignUuid	point	unsignedSmallInteger
geometryCollection	polygon	unsignedTinyInteger
geometry	rememberToken	ulidMorphs
id	set	uuidMorphs
increments	smallIncrements	ulid
integer	smallInteger	uuid
ipAddress	softDeletesTz	year
json	softDeletes	

Creating Indexes

The Laravel schema builder supports several types of indexes. The following example creates a new `email` column and specifies that its values should be unique. To create the index, we can chain the `unique` method onto the column definition:

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

Schema::table('users', function (Blueprint $table) {
    $table->string('email')->unique();
});
```

```
$table->unique('email');
```

```
$table->index(['account_id', 'created_at']);
```

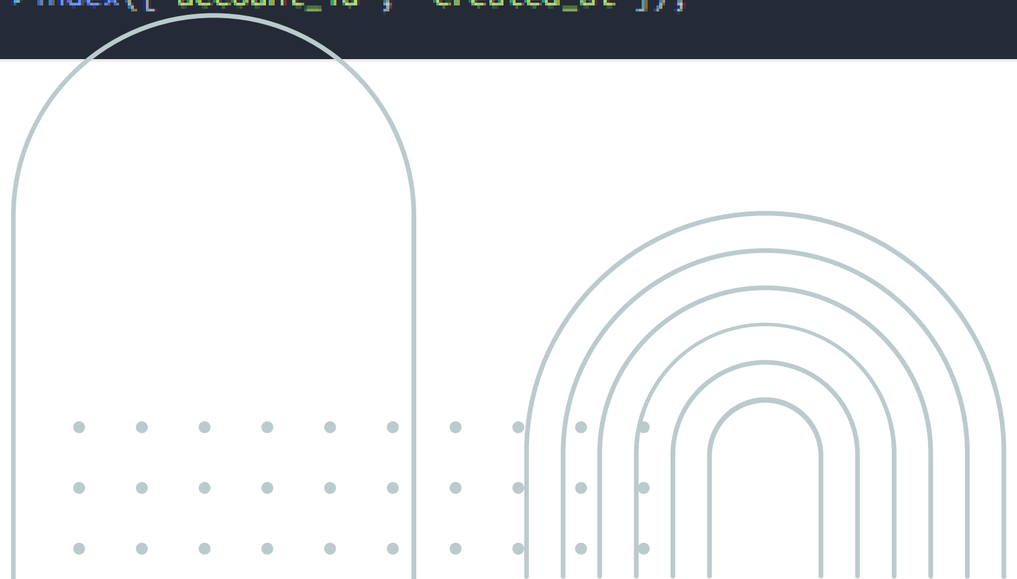
Foreign Key Constraints

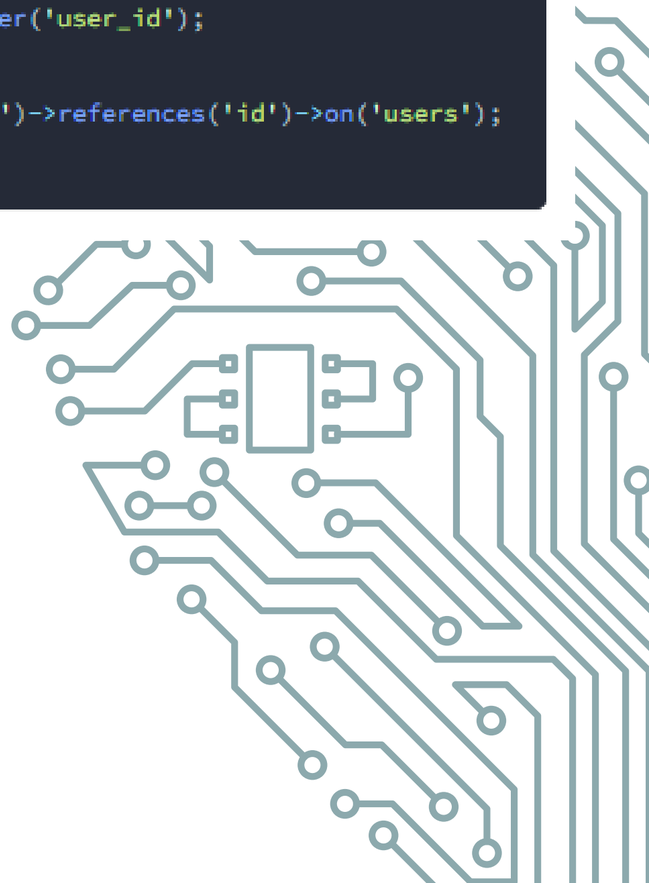
Laravel also provides support for creating foreign key constraints, which are used to force referential integrity at the database level. For example, let's define a `user_id` column on the `posts` table that references the `id` column on a `users` table:

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

Schema::table('posts', function (Blueprint $table) {
    $table->unsignedBigInteger('user_id');

    $table->foreign('user_id')->references('id')->on('users');
});
```





LATIHAN

Buat 2 buah tabel dengan struktur seperti berikut menggunakan fitur migration pada laravel !