DATABASE: MIGRATIONS

1.1. Introduction

Migration giống như một hệ thống quản lý phiên bản như Git nhưng dành cho database. Migration cho phép định nghĩa định nghĩa, cập nhật thay đổi các table trong Database. Đồng thời các thao tác với Database này còn có thể sử dụng trên các loại Database khác nhau như MySQL, SQL Server, Postgres, … mà không cần phải chỉnh sửa lại code theo Database sử dụng.

Điều kiện tiên quyết để chạy migration một cách thành công:

- Phải có kết nối với database;
- Migrations muốn sử dụng được thì phải nằm trong thư mục App\database\migrations.

1.2. Generating Migrations

Sử dụng lệnh `make:migration` Artisan để tạo một database migration. Migration mới sẽ được đặt trong thư mục `database/migrations`. Mỗi migration filename chứa một timestamp cho phép Laravel xác định thứ tự của migrations:

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

Laravel sẽ dựa vào tên của migration để đặt tên của bảng và xem xét việc có tạo một bảng mới hay không. Nếu Laravel có thể xác định tên bảng từ tên migration, Laravel sẽ điền trước file migration đã tạo với bảng được chỉ định. Nếu không, có thể chỉ định bảng trong file migration theo cách thủ công.

Nếu muốn chỉ định một đường dẫn tùy chỉnh cho migration đã tạo, sử dụng tùy chọn `path` khi thực hiện lệnh `make:migration`. Đường dẫn đã cho phải liên quan đến đường dẫn cơ sở của ứng dụng.

1.2.1. Squashing Migrations

Khi xây dựng ứng dụng, có thể tích lũy ngày càng nhiều migrations theo thời gian. Điều này có thể dẫn đến thư mục `database/migrations` trở nên cồng kềnh với hàng trăm migrations. Nếu muốn, có thể "squash" migrations vào một file SQL duy nhất, bằng cách thực hiện lệnh `schema:dump`:

```
php artisan schema:dump
```

Trong trường hợp vừa muốn squash vào file và xóa luôn hết migration hiện có thì bạn có thể sử dụng thêm tham số `--prune`:

```
// Dump the current database schema and prune all existing migrations...

php artisan schema:dump --prune
```

Khi thực thi lệnh này, Laravel sẽ ghi một file "chema" vào thư mục `database/schema` của ứng dụng. Bây

giờ, khi migrate cơ sở dữ liệu và không có migrations nào khác được thực hiện, Laravel sẽ thực thi các câu lệnh SQL của file lược đồ trước.

Migration squashing chỉ khả dụng cho cơ sở dữ liệu MySQL, PostgreSQL và SQLite và sử dụng database's command-lineu. Schema dumps có thể không được khôi phục vào cơ sở dữ liệu SQLite trong bộ nhớ.

1.3. Migration Structure

Một lớp migration chứa hai phương thức: `up` và `down`. Phương thức `up` được sử dụng để thêm bảng, cột hoặc chỉ mục mới vào cơ sở dữ liệu, trong khi phương thức `down` sẽ đảo ngược các thao tác được thực hiện bởi phương thức `up`.

Trong cả hai phương pháp này, có thể sử dụng Laravel schema Builder để tạo và sửa đổi các bảng một cách rõ ràng. Để tìm hiểu về các phương thức có sẵn trên `Schema`, hãy check out its documentation.

Ví dụ, migration sau đây tạo ra một bảng `flights`:

```php
<?php

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

class CreateFlightsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('flights', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('airline');
            $table->timestamps();
        });
```

```
    }

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

Anonymous Migrations

Trong ví dụ trên, Laravel sẽ tự động gán tên lớp cho tất cả các migrations tạo bằng lệnh `make:migration`. Tuy nhiên, nếu muốn có thể trả về một lớp ẩn danh (anonymous class) từ file migration. Điều này chủ yếu hữu ích nếu ứng dụng tích lũy nhiều migrations và hai trong số chúng có xung đột tên lớp (class name collision):

```
<?php

use Illuminate\Database\Migrations\Migration;

return new class extends Migration
{
    //
};
```

Setting The Migration Connection

Nếu migration tương tác với kết nối cơ sở dữ liệu khác với kết nối cơ sở dữ liệu mặc định của ứng dụng, nên đặt thuộc tính `$connection` của migration:

```
/**
 * The database connection that should be used by the migration.
```

```
     *
     * @var string
     */
    protected $connection = 'pgsql';


    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        //
    }
```

1.4. Running Migrations

Để chạy tất cả các migrations chưa hoàn thành, hãy thực hiện lệnh Artisan `migrate`:

```
php artisan migrate
```

Nếu muốn xem những migrations nào đã chạy, có thể sử dụng lệnh Artisan `migrate:status`:

```
php artisan migrate:status
```

Forcing Migrations To Run In Production

Một số hoạt động migration có tính chất phá hoại, có nghĩa là chúng có thể khiến mất dữ liệu. Để bảo vệ khỏi việc chạy các lệnh này với cơ sở dữ liệu, sẽ được nhắc xác nhận trước khi các lệnh được thực thi. Để buộc các lệnh chạy mà không có lời nhắc, hãy sử dụng cờ `--force`:

```
php artisan migrate --force
```

1.4.1. Rolling Back Migrations

Để roll back hoạt động migration mới nhất, có thể sử dụng lệnh Artisan `rollback`. Lệnh này quay trở lại "batch" migrations cuối cùng, có thể bao gồm nhiều file migration:

```
php artisan migrate:rollback
```

Có thể roll back một số lần hạn chế của migrations bằng cách cung cấp tùy chọn `step` cho lệnh `rollback`.

Ví dụ: lệnh sau sẽ roll back lại năm lần migrations cuối cùng:

```
php artisan migrate:rollback --step=5
```

Lệnh `migrate:reset` sẽ roll back tất cả các migrations ứng dụng

```
php artisan migrate:reset
```

Roll Back & Migrate Using A Single Command

Lệnh `migrate:refresh` sẽ roll back tất cả các migrations và sau đó thực hiện lệnh `migrate`. Lệnh này tạo lại toàn bộ cơ sở dữ liệu một cách hiệu quả:

```
php artisan migrate:refresh


// Refresh the database and run all database seeds...
php artisan migrate:refresh --seed
```

Có thể roll back và re-migrate một số lần hạn chế migrations bằng cách cung cấp tùy chọn `step` cho lệnh `refresh`.

Ví dụ, lệnh sau sẽ roll back và re-migrate lại năm lần migrations cuối cùng:

```
php artisan migrate:refresh --step=5
```

Drop All Tables & Migrate

Lệnh `migrate:fresh` sẽ drop tất cả các bảng khỏi cơ sở dữ liệu và sau đó thực thi lệnh `migrate`:

```
php artisan migrate:fresh


php artisan migrate:fresh --seed
```

1.5. Tables

1.5.1. Creating Tables

Để tạo một bảng cơ sở dữ liệu mới, sử dụng phương thức `create` trên `Schema` facade. Phương thức `create` chấp nhận hai đối số: đối số đầu tiên là tên của bảng, trong khi đối số thứ hai là một closure nhận một đối tượng `Blueprint` có thể được sử dụng để xác định bảng mới:

```
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();
});
```

Khi tạo bảng, có thể sử dụng bất kỳ phương thức schema builder's column methods để xác định các cột của bảng.

Checking For Table / Column Existence

Có thể kiểm tra sự tồn tại của một bảng hoặc cột bằng các phương thức `hasTable` và `hasColumn`:

```
if (Schema::hasTable('users')) {
    // The "users" table exists...
}

if (Schema::hasColumn('users', 'email')) {
    // The "users" table exists and has an "email" column...
}
```

Database Connection & Table Options

Nếu muốn thực hiện thao tác schema trên kết nối cơ sở dữ liệu không phải là kết nối mặc định của ứng dụng, hãy sử dụng phương thức `connection`:

```
Schema::connection('sqlite')->create('users', function (Blueprint $table) {
    $table->id();
});
```

Ngoài ra, một số thuộc tính và phương thức khác có thể được sử dụng để xác định các khía cạnh khác của việc tạo bảng. Thuộc tính `engine` có thể được sử dụng để chỉ định table's storage engine khi sử dụng MySQL:

```
Schema::create('users', function (Blueprint $table) {
    $table->engine = 'InnoDB';

    // ...
});
```

Các thuộc tính `charset` và `collation` có thể được sử dụng để chỉ định character set và collation cho bảng đã tạo khi sử dụng MySQL:

```
Schema::create('users', function (Blueprint $table) {
    $table->charset = 'utf8mb4';
    $table->collation = 'utf8mb4_unicode_ci';

    // ...
});
```

Phương thức `temporary` được sử dụng để chỉ ra rằng bảng phải là "temporary". Temporary tables chỉ hiển thị với phiên cơ sở dữ liệu của kết nối hiện tại và tự động bị loại bỏ khi kết nối bị đóng:

```
Schema::create('calculations', function (Blueprint $table) {
    $table->temporary();

    // ...
});
```

1.5.2. Updating Tables

Phương thức `table` trên `Schema` facade được sử dụng để cập nhật các bảng hiện có. Giống như phương thức `create`, phương thức `table` chấp nhận hai đối số: tên của bảng và một closure nhận một instance `Blueprint` có thể sử dụng để thêm cột hoặc chỉ mục vào bảng:

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

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

1.5.3. Renaming / Dropping Tables

Để đổi tên một bảng cơ sở dữ liệu hiện có, hãy sử dụng phương thức `rename`:

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

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

Để drop một bảng hiện có, sử dụng các phương thức `drop` hoặc `dropIfExists`:

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

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

Renaming Tables With Foreign Keys

Trước khi đổi tên bảng, nên xác minh rằng bất kỳ ràng buộc khóa ngoại nào trên bảng đều có tên rõ ràng trong file migration thay vì để Laravel gán tên dựa trên quy ước. Nếu không, tên ràng buộc khóa ngoại sẽ tham chiếu đến tên bảng cũ.

1.6. Columns

1.6.1. Creating Columns

Phương thức `table` trên `Schema` facade được sử dụng để cập nhật các bảng hiện có. Giống như phương thức `create`, phương thức `table` chấp nhận hai đối số: tên của bảng và một closure nhận một instance `Illuminate\Database\Schema\Blueprint` có thể sử dụng để thêm cột vào bảng:

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

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

1.6.2. Available Column Types

Schema Builder Blueprint cung cấp nhiều phương thức tương ứng với các loại cột khác nhau có thể thêm vào các bảng cơ sở dữ liệu. Mỗi phương thức có sẵn được liệt kê trong bảng dưới đây:

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	nullableUuidMorphs	unsignedInteger
foreignUuid	point	unsignedMediumInteger
geometryCollection	polygon	unsignedSmallInteger
geometry	rememberToken	unsignedTinyInteger
id	set	uuidMorphs
increments	smallIncrements	uuid
integer	smallInteger	year
ipAddress	softDeletesTz	
json	softDeletes	

bigIncrements()

Phương thức `bigIncrements` tạo cột tương đương `UNSIGNED BIGINT` (khóa chính) tự động tăng dần:

```
$table->bigIncrements('id');
```

bigInteger()

Phương thức `bigInteger` tạo một cột tương đương `BIGINT`:

```
$table->bigInteger('votes');
```

binary()

Phương thức `binary` tạo một cột tương đương `BLOB`:

```
$table->binary('photo');
```

boolean()

Phương thức `boolean` tạo một cột tương đương `BOOLEAN`:

```
$table->boolean('confirmed');
```

char()

Phương thức `char` tạo một cột tương đương `CHAR` có độ dài cho trước:

```
$table->char('name', 100);
```

dateTimeTz()

Phương thức `dateTimeTz` tạo cột tương đương `DATETIME` (with timezone) với độ chính xác tùy chọn (total digits):

```
$table->dateTimeTz('created_at', $precision = 0);
```

dateTime()

Phương thức `dateTime` tạo cột tương đương `DATETIME` với độ chính xác tùy chọn (total digits):

```
$table->dateTime('created_at', $precision = 0);
```

date()

Phương thức `date` tạo một cột tương đương `DATE`:

```
$table->date('created_at');
```

decimal()

Phương pháp `decimal` tạo ra một cột tương đương `DECIMAL` với độ chính xác đã cho (total digits) và scale (decimal digits):

```
$table->decimal('amount', $precision = 8, $scale = 2);
```

double()

Phương pháp k `double` tạo ra một cột tương đương `DOUBLE` với độ chính xác đã cho (total digits) và scale (decimal digits):

```
$table->double('amount', 8, 2);
```

enum()

Phương thức `enum` tạo một cột tương đương `ENUM` với các giá trị hợp lệ đã cho:

```
$table->enum('difficulty', ['easy', 'hard']);
```

float()

Phương thức `float` tạo ra một cột tương đương `FLOAT` với độ chính xác đã cho (total digits) và scale (decimal digits):

```
$table->float('amount', 8, 2);
```

foreignId()

Phương thức `foreignId` tạo một cột tương đương `UNSIGNED BIGINT`:

```
$table->foreignId('user_id');
```

foreignIdFor()

Phương thức `foreignIdFor` thêm một cột tương đương `{column}_id UNSIGNED BIG INT` cho một model class:

```
$table->foreignIdFor(User::class);
```

foreignUuid()

Phương thức `foreignUuid` tạo một cột `UUID` tương đương:

```
$table->foreignUuid('user_id');
```

geometryCollection()

Phương thức `geometryCollection` tạo ra một cột tương đương `GEOMETRYCOLLECTION`:

```
$table->geometryCollection('positions');
```

geometry()

Phương thức `geometry` tạo ra một cột tương đương `GEOMETRY`:

```
$table->geometry('positions');
```

id()

Phương thức `id` là một bí danh (alias) của phương thức `bigIncrements`. Theo mặc định, phương thức sẽ tạo một cột `id`; tuy nhiên, có thể truyền một tên cột nếu muốn gán một tên khác cho cột:

```
$table->id();
```

increments()

Phương thức `increments` tạo một cột tương đương `UNSIGNED INTEGER` tự động tăng dần làm khóa chính (primary key):

```
$table->increments('id');
```

integer()

Phương thức `integer` tạo một cột tương đương `INTEGER`:

```
$table->integer('votes');
```

ipAddress()

Phương thức `ipAddress` tạo một cột tương đương `VARCHAR`:

```
$table->ipAddress('visitor');
```

json()

Phương thức `json` tạo một cột tương đương `JSON`:

```
$table->json('options');
```

jsonb()

Phương thức `jsonb` tạo một cột tương đương `JSONB`:

```
$table->jsonb('options');
```

lineString()

Phương thức `lineString` tạo một cột tương đương `LINESTRING`:

```
$table->lineString('positions');
```

longText()

Phương thức `longText` tạo một cột tương đương `LONGTEXT`:

```
$table->longText('description');
```

macAddress()

Phương thức `macAddress` tạo một cột dùng để chứa địa chỉ MAC. Một số hệ thống cơ sở dữ liệu, chẳng hạn như PostgreSQL, có kiểu cột dành riêng cho kiểu dữ liệu này. Các hệ thống cơ sở dữ liệu khác sẽ sử dụng một cột tương đương chuỗi:

```
$table->macAddress('device');
```

mediumIncrements()

Phương thức `mediumIncrements` tạo cột tương đương `UNSIGNED MEDIUMINT` tự động tăng dần làm khóa chính (primary key):

```
$table->mediumIncrements('id');
```

mediumInteger()

Phương thức `mediumInteger` tạo một cột tương đương `MEDIUMINT`:

```
$table->mediumInteger('votes');
```

mediumText()

Phương thức `mediumText` tạo một cột tương đương `MEDIUMTEXT`:

```
$table->mediumText('description');
```

morphs()

Phương thức `morphs` là một phương thức tiện lợi thêm một cột tương đương `{column}_id` `UNSIGNED BIGINT` và một cột tương đương `{column}_type` `VARCHAR`.

Phương thức này được sử dụng khi xác định các cột cần thiết cho polymorphic Eloquent relationship.

Ví dụ sau, các cột `taggable_id` và `taggable_type` sẽ được tạo:

```
$table->morphs('taggable');
```

multiLineString()

Phương thức `multiLineString` tạo một cột tương đương `MULTILINESTRING`:

```
$table->multiLineString('positions');
```

multiPoint()

Phương thức `multiPoint` tạo một cột tương đương `MULTIPOINT`:

```
$table->multiPoint('positions');
```

multiPolygon()

Phương thức `multiPolygon` tạo một cột tương đương `MULTIPOLYGON`:

```
$table->multiPolygon('positions');
```

nullableTimestamps()

Phương thức `nullableTimestamps` là một bí danh (alias) của phương thức timestamps:

```
$table->nullableTimestamps(0);
```

nullableMorphs()

Phương thức này tương tự như phương pháp morphs; tuy nhiên, các cột được tạo sẽ là "nullable":

```
$table->nullableMorphs('taggable');
```

nullableUuidMorphs()

Phương thức này tương tự như phương thức uuidMorphs; tuy nhiên, các cột được tạo sẽ là "nullable":

```
$table->nullableUuidMorphs('taggable');
```

point()

Phương thức `point` tạo một cột tương đương `POINT`:

```
$table->point('position');
```

polygon()

Phương thức `polygon` tạo một cột tương đương `POLYGON`:

```
$table->polygon('position');
```

rememberToken()

Phương thức rememberToken tạo ra một cột tương đương VARCHAR(100) có thể null, nhằm mục đích lưu trữ "remember me" authentication token hiện tại:

```
$table->rememberToken();
```

set()

Phương thức `set` tạo một cột tương đương `SET` với danh sách các giá trị hợp lệ đã cho:

```
$table->set('flavors', ['strawberry', 'vanilla']);
```

smallIncrements()

Phương thức `smallIncrements` tạo một cột tương đương `UNSIGNED SMALLINT` tự động tăng dần làm khóa chính (primary key):

```
$table->smallIncrements('id');
```

smallInteger()

Phương thức `smallInteger` tạo một cột tương đương `SMALLINT`:

```
$table->smallInteger('votes');
```

softDeletesTz()

Phương thức `softDeletesTz` thêm cột nullable tương đương với `deleted_at` `TIMESTAMP` (with timezone) với độ chính xác tùy chọn (total digits). Cột này nhằm lưu trữ dấu thời gian `deleted_at` cần thiết cho chức năng "soft delete" của Eloquent:

```
$table->softDeletesTz($column = 'deleted_at', $precision = 0);
```

softDeletes()

Phương thức `softDeletes` thêm cột nullable tương đương `deleted_at` `TIMESTAMP` với độ chính xác tùy chọn (total digits). Cột này nhằm lưu trữ dấu thời gian `deleted_at` cần thiết cho chức năng "soft delete" của Eloquent:

```
$table->softDeletes($column = 'deleted_at', $precision = 0);
```

string()

Phương thức `string` tạo một cột tương đương `VARCHAR` có độ dài đã cho:

```
$table->string('name', 100);
```

text()

Phương thức `text` tạo một cột tương đương `TEXT`:

```
$table->text('description');
```

timeTz()

Phương thức `timeTz` tạo cột tương đương `TIME` (with timezone) với độ chính xác tùy chọn (total digits):

```
$table->timeTz('sunrise', $precision = 0);
```

time()

Phương thức `time` tạo một cột tương đương `TIME` với độ chính xác tùy chọn (total digits):

```
$table->time('sunrise', $precision = 0);
```

timestampTz()

Phương thức `timestampTz` tạo cột tương đương `TIMESTAMP` (with timezone) với độ chính xác tùy chọn (total digits):

```
$table->timestampTz('added_at', $precision = 0);
```

timestamp()

Phương thức `timestamp` tạo cột tương đương `TIMESTAMP` với độ chính xác tùy chọn (total digits):

```
$table->timestamp('added_at', $precision = 0);
```

timestampsTz()

Phương thức `timestampsTz` tạo các cột tương đương với `created_at` và `updated_at` `TIMESTAMP` (with timezone) với độ chính xác tùy chọn (total digits):

```
$table->timestampsTz($precision = 0);
```

timestamps()

Phương thức `timestamps` tạo các cột tương đương với `created_at` và `updated_at` `TIMESTAMP` với độ chính xác tùy chọn (total digits):

```
$table->timestamps($precision = 0);
```

tinyIncrements()

Phương thức `tinyIncrements` tạo cột tương đương `UNSIGNED TINYINT` tự động tăng dần làm khóa chính (primary key):

```
$table->tinyIncrements('id');
```

tinyInteger()

Phương thức `tinyInteger` tạo một cột tương đương `TINYINT`:

```
$table->tinyInteger('votes');
```

tinyText()

Phương thức `tinyText` tạo một cột tương đương `TINYTEXT`:

```
$table->tinyText('notes');
```

unsignedBigInteger()

Phương thức `unsignedBigInteger` tạo một cột tương đương `UNSIGNED BIGINT`:

```
$table->unsignedBigInteger('votes');
```

unsignedDecimal()

Phương thức `unsignedDecimal` tạo cột tương đương `UNSIGNED DECIMAL` với độ chính xác tùy chọn (total digits) và scale (decimal digits):

```
$table->unsignedDecimal('amount', $precision = 8, $scale = 2);
```

unsignedInteger()

Phương thức `unsignedInteger` tạo một cột tương đương `UNSIGNED INTEGER`:

```
$table->unsignedInteger('votes');
```

unsignedMediumInteger()

Phương thức `unsignedMediumInteger` tạo một cột tương đương `UNSIGNED MEDIUMINT`:

```
$table->unsignedMediumInteger('votes');
```

unsignedSmallInteger()

Phương thức `unsignedSmallInteger` tạo một cột tương đương `UNSIGNED SMALLINT`:

```
$table->unsignedSmallInteger('votes');
```

unsignedTinyInteger()

Phương thức `unsignedTinyInteger` tạo một cột tương đương `UNSIGNED TINYINT`:

```
$table->unsignedTinyInteger('votes');
```

uuidMorphs()

Phương thức `uuidMorphs` là một phương thức tiện lợi bổ sung một cột tương đương `column}_id CHAR(36)` và một cột tương đương `{column}_type VARCHAR`.

Phương thức này được sử dụng khi xác định các cột cần thiết cho polymorphic Eloquent relationship sử dụng các định danh UUID.

ví dụ, các cột `taggable_id` và `taggable_type` sẽ được tạo:

```
$table->uuidMorphs('taggable');
```

uuid()

Phương thức `uuid` tạo một cột tương đương `UUID`:

```
$table->uuid('id');
```

year()

Phương pháp `year` tạo một cột tương đương `YEAR`:

```
$table->year('birth_year');
```

1.6.3. Column Modifiers

Ngoài các loại cột được liệt kê ở trên, một số column "modifiers" có thể sử dụng khi thêm cột vào bảng cơ sở dữ liệu.

Ví dụ, để làm cho cột là "nullable", có thể sử dụng phương thức `nullable`:

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

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

Bảng sau đây chứa tất cả các column modifiers có sẵn. Danh sách này không bao gồm các Index modifiers:

Modifier	Description
`->after('column')`	Place the column "after" another column (MySQL).
`->autoIncrement()`	Set INTEGER columns as auto-incrementing (primary key).

Modifier	Description
`->charset('utf8mb4')`	Specify a character set for the column (MySQL).
`->collation('utf8mb4_unicode_ci')`	Specify a collation for the column (MySQL/PostgreSQL/SQL Server).
`->comment('my comment')`	Add a comment to a column (MySQL/PostgreSQL).
`->default($value)`	Specify a "default" value for the column.
`->first()`	Place the column "first" in the table (MySQL).
`->from($integer)`	Set the starting value of an auto-incrementing field (MySQL / PostgreSQL).
`->nullable($value = true)`	Allow NULL values to be inserted into the column.
`->storedAs($expression)`	Create a stored generated column (MySQL / PostgreSQL).
`->unsigned()`	Set INTEGER columns as UNSIGNED (MySQL).
`->useCurrent()`	Set TIMESTAMP columns to use CURRENT_TIMESTAMP as default value.
`->useCurrentOnUpdate()`	Set TIMESTAMP columns to use CURRENT_TIMESTAMP when a record is updated.
`->virtualAs($expression)`	Create a virtual generated column (MySQL).
`->generatedAs($expression)`	Create an identity column with specified sequence options (PostgreSQL).
`->always()`	Defines the precedence of sequence values over input for an identity column (PostgreSQL).

Default Expressions

`Default` modifier chấp nhận một giá trị hoặc một instance l `Illuminate\Database\Query\Expression`. Việc sử dụng `Expression` instance sẽ ngăn Laravel gói giá trị trong dấu ngoặc kép và cho phép sử dụng các chức năng cụ thể của cơ sở dữ liệu. Một tình huống mà điều này đặc biệt hữu ích là khi cần gán giá trị mặc định cho các cột JSON:

```
<?php

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

```
use Illuminate\Database\Migrations\Migration;


class CreateFlightsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('flights', function (Blueprint $table) {
            $table->id();
            $table->json('movies')->default(new Expression('(JSON_ARRAY())'));
            $table->timestamps();
        });
    }
}
```

Column Order

Khi sử dụng cơ sở dữ liệu MySQL, phương thức `after` có thể được sử dụng để thêm các cột sau một cột hiện có trong lược đồ (schema):

```
$table->after('password', function ($table) {
    $table->string('address_line1');
    $table->string('address_line2');
    $table->string('city');
});
```

1.6.4. Modifying Columns

Prerequisites

Trước khi sửa đổi một cột, phải cài đặt package `doctrine/dbal` bằng Composer package manager. Thư viện Doctrine DBAL được sử dụng để xác định trạng thái hiện tại của cột và tạo các truy vấn SQL cần thiết để thực hiện các thay đổi được yêu cầu đối với cột:

```
composer require doctrine/dbal
```

Nếu định sửa đổi các cột được tạo bằng phương thức `timestamp`, cũng phải thêm cấu hình sau vào file cấu hình `config/database.php` của ứng dụng:

```
use Illuminate\Database\DBAL\TimestampType;


'dbal' => [
    'types' => [
        'timestamp' => TimestampType::class,
    ],
],
```

Lưu ý: Nếu ứng dụng đang sử dụng Microsoft SQL Server, hãy chắc rằng đã cài đặt `doctrine/dbal:^3.0`.

Updating Column Attributes

Phương thức `change` cho phép bạn sửa đổi loại và thuộc tính của các cột hiện có. Ví dụ, muốn tăng kích thước của một `string` column (cột tên từ 25 lên 50), chỉ cần xác định trạng thái mới của cột và sau đó gọi phương thức `change`:

```
Schema::table('users', function (Blueprint $table) {
    $table->string('name', 50)->change();
});
```

Cũng có thể sửa đổi một cột để có thể nullable:

```
Schema::table('users', function (Blueprint $table) {
    $table->string('name', 50)->nullable()->change();
});
```

Có thể modify các loại cột sau: `bigInteger`, `binary`, `boolean`, `date`, `dateTime`, `dateTimeTz`, `decimal`, `integer`, `json`, `longText`, `mediumText`, `smallInteger`, `string`, `text`, `time`, `unsignedBigInteger`, `unsignedInteger`, `unsignedSmallInteger`, và `uuid`. Để modify một `timestamp` column type, phải Doctrine type must be registered.

Renaming Columns

Để đổi tên một cột, có thể sử dụng phương thức `renameColumn` được cung cấp bởi Schema Builder

Blueprint. Trước khi đổi tên cột, hãy đảm bảo rằng đã cài đặt thư viện `doctrine/dbal` thông qua Composer package manager:

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

Lưu ý: Đổi tên cột `enum` hiện không được hỗ trợ.

1.6.5. Dropping Columns

Để drop một cột, sử dụng phương thức `dropColumn` trên Schema Builder Blueprint. Nếu ứng dụng đang sử dụng cơ sở dữ liệu SQLite, phải cài đặt gói `doctrine/dbal` thông qua Composer package manager trước khi phương thức `dropColumn` có thể được sử dụng:

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

Có thể drop nhiều cột từ một bảng bằng cách truyền một mảng tên cột vào phương thức `dropColumn`:

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

Lứ ý: Dropping hoặc modifying nhiều cột trong một single migration trong khi sử dụng cơ sở dữ liệu SQLite không được hỗ trợ.

Available Command Aliases

Laravel cung cấp một số phương thức liên quan đến việc dropping các loại cột phổ biến. Các phương thức này được mô tả trong bảng dưới đây:

Command	Description
`$table->dropMorphs('morphable');`	Drop the `morphable_id` and `morphable_type` columns.
`$table->dropRememberToken();`	Drop the `remember_token` column.
`$table->dropSoftDeletes();`	Drop the `deleted_at` column.

Command	Description
`$table->dropSoftDeletesTz();`	Alias of `dropSoftDeletes()` method.
`$table->dropTimestamps();`	Drop the `created_at` and `updated_at` columns.
`$table->dropTimestampsTz();`	Alias of `dropTimestamps()` method.

1.7. Indexes

1.7.1. Creating Indexes

Laravel Schema Builder hỗ trợ một số loại chỉ mục. Ví dụ sau tạo một cột `email` mới và chỉ định rằng các giá trị của nó phải là duy nhất. Để tạo chỉ mục, thêm phương thức `unique` vào định nghĩa cột:

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

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

Ngoài ra, có thể tạo chỉ mục sau khi xác định cột. Để làm như vậy, gọi phương thức `unique` trên Schema Builder Blueprint. Phương thức này chấp nhận tên của cột sẽ nhận được một chỉ mục duy nhất:

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

Thậm chí có thể truyền một mảng cột đến một phương thức `index` để tạo một chỉ mục phức hợp (hoặc kết hợp):

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

Khi tạo chỉ mục, Laravel sẽ tự động tạo tên chỉ mục dựa trên bảng, tên cột và kiểu chỉ mục, nhưng có thể truyền đối số thứ hai vào phương thức để tự chỉ định tên chỉ mục:

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

Available Index Types

Lớp Schema Builder Blueprint của Laravel cung cấp các phương thức để tạo từng loại chỉ mục được hỗ trợ bởi Laravel. Mỗi phương thức chỉ mục chấp nhận một đối số thứ hai tùy chọn để chỉ định tên của chỉ mục.

Nếu bị bỏ qua, tên sẽ được lấy từ tên của bảng và (các) cột được sử dụng cho chỉ mục, cũng như kiểu chỉ mục. Các phương thức chỉ mục có sẵn được mô tả trong bảng dưới đây:

Command	Description
`$table->primary('id');`	Adds a primary key.
`$table->primary(['id', 'parent_id']);`	Adds composite keys.
`$table->unique('email');`	Adds a unique index.
`$table->index('state');`	Adds an index.
`$table->spatialIndex('location');`	Adds a spatial index (except SQLite).

Index Lengths & MySQL / MariaDB

Theo mặc định, Laravel sử dụng bộ ký tự `utf8mb4`. Nếu đang chạy phiên bản MySQL cũ hơn bản phát hành 5.7.7 hoặc MariaDB cũ hơn bản phát hành 10.2.2, có thể cần phải định cấu hình theo cách thủ công độ dài chuỗi mặc định được tạo bởi migrations để MySQL tạo chỉ mục. Có thể định cấu hình độ dài chuỗi mặc định bằng cách gọi phương thức `Schema::defaultStringLength` trong phương thức `boot` của lớp `App\Providers\AppServiceProvider`:

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

/**
 * Bootstrap any application services.
 *
 * @return void
 */
public function boot()
{
    Schema::defaultStringLength(191);
}
```

Ngoài ra, có thể bật tùy chọn `innodb_large_prefix` cho cơ sở dữ liệu. Tham khảo tài liệu của cơ sở dữ liệu để biết hướng dẫn về cách bật tùy chọn này đúng cách.

1.7.2. Renaming Indexes

Để đổi tên chỉ mục, có thể sử dụng phương thức `renameIndex` được cung cấp bởi Schema Builder Blueprint. Phương thức này chấp nhận tên chỉ mục hiện tại làm đối số đầu tiên và tên mong muốn làm đối số thứ hai:

```
$table->renameIndex('from', 'to')
```

1.7.3. Dropping Indexes

Để bỏ một chỉ mục, phải chỉ định tên của chỉ mục. Theo mặc định, Laravel tự động gán tên chỉ mục dựa trên tên bảng, tên của cột được lập chỉ mục và kiểu chỉ mục. Dưới đây là một số ví dụ:

Command	Description
`$table->dropPrimary('users_id_primary');`	Drop a primary key from the "users" table.
`$table->dropUnique ('users_email_unique');`	Drop a unique index from the "users" table.
`$table->dropIndex('geo_state_index');`	Drop a basic index from the "geo" table.
`$table->dropSpatialIndex ('geo_location_spatialindex');`	Drop a spatial index from the "geo" table (except SQLite).

Nếu truyền một mảng cột vào một phương thức làm giảm chỉ mục, tên chỉ mục thông thường sẽ được tạo dựa trên tên bảng, cột và kiểu chỉ mục:

```
Schema::table('geo', function (Blueprint $table) {
    $table->dropIndex(['state']); // Drops index 'geo_state_index'
});
```

1.7.4. Foreign Key Constraints

Laravel cũng cung cấp hỗ trợ để tạo các ràng buộc khóa ngoại, được sử dụng để buộc tính toàn vẹn tham chiếu ở cấp cơ sở dữ liệu. Ví dụ, hãy xác định cột `user_id` trên bảng `posts` tham chiếu đến cột `id` trên bảng `users`:

```
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');
});
```

Vì cú pháp này khá dài dòng, Laravel cung cấp các phương thức bổ sung, ngắn gọn hơn sử dụng các quy ước để cung cấp trải nghiệm tốt hơn cho nhà phát triển. Khi sử dụng phương thức `foreignId` để tạo cột, ví dụ trên có thể được viết lại như sau:

```
Schema::table('posts', function (Blueprint $table) {
    $table->foreignId('user_id')->constrained();
});
```

Phương thức `foreignId` tạo ra một cột tương đương `UNSIGNED BIGINT`, trong khi phương thức `constrained` sẽ sử dụng các quy ước để xác định bảng và tên cột đang được tham chiếu. Nếu tên bảng không khớp với các quy ước của Laravel, có thể chỉ định tên bảng bằng cách truyền nó làm đối số cho phương thức `constrained`:

```
Schema::table('posts', function (Blueprint $table) {
    $table->foreignId('user_id')->constrained('users');
});
```

Cũng có thể chỉ định hành động mong muốn cho các thuộc tính "on delete" và "on update" của ràng buộc:

```
$table->foreignId('user_id')
      ->constrained()
      ->onUpdate('cascade')
      ->onDelete('cascade');
```

Bất kỳ column modifiers nào phải được gọi trước phương thức `constrained`:

```
$table->foreignId('user_id')
      ->nullable()
      ->constrained();
```

Dropping Foreign Keys

Để xóa khóa ngoại, sử dụng phương thức `dropForeign`, truyền tên của ràng buộc khóa ngoại sẽ bị xóa làm đối số. Các ràng buộc khóa ngoại sử dụng quy ước đặt tên giống như các chỉ mục. Nói cách khác, tên ràng buộc khóa ngoại dựa trên tên của bảng và các cột trong ràng buộc, theo sau là hậu tố "_foreign":

```
$table->dropForeign('posts_user_id_foreign');
```

Ngoài ra, có thể truyền một mảng chứa tên cột có khóa ngoại vào phương thức `dropForeign`. Mảng sẽ được chuyển đổi thành tên ràng buộc khóa ngoại bằng cách sử dụng các quy ước đặt tên cho ràng buộc của Laravel:

```
$table->dropForeign(['user_id']);
```

Toggling Foreign Key Constraints

Có thể bật hoặc tắt các ràng buộc khóa ngoại trong migrations bằng cách sử dụng các phương pháp sau:

```
Schema::enableForeignKeyConstraints();

Schema::disableForeignKeyConstraints();
```

SQLite vô hiệu hóa các ràng buộc khóa ngoại theo mặc định. Khi sử dụng SQLite, hãy đảm bảo bật enable foreign key support cơ sở dữ liệu trước khi cố gắng tạo chúng trong quá trình migrations. Ngoài ra, SQLite chỉ hỗ trợ khóa ngoại khi tạo bảng và không hỗ trợ khi bảng bị thay đổi (not when tables are altered).

1.8. Events

Để thuận tiện, mỗi thao tác migration gửi một event. Tất cả các events sau đây mở rộng lớp `Illuminate\Database\Events\MigrationEvent`:

Class	Description
`Illuminate\Database\Events\MigrationsStarted`	A batch of migrations is about to be executed.
`Illuminate\Database\Events\MigrationsEnded`	A batch of migrations has finished executing.
`Illuminate\Database\Events\MigrationStarted`	A single migration is about to be executed.
`Illuminate\Database\Events\MigrationEnded`	A single migration has finished executing.