Создание миграций

Миграции похожи на систему контроля версий, но только для базы данных. Они позволяют команде разработчиков легко изменять схему БД приложения и делиться этими изменениями.
Миграции обычно связаны с построителем схем Laravel (schema builder) для облегчения создания схемы БД приложения.

Создать миграцию можно с помощью следующей команды

```
$ php artisan make:migration create_posts_table
```

В результате в папке **database/migrations** будет создан файл **xxx_create_posts_table.php**

В файле содержится класс и два метода:

- **up** – действия, которые применяются к БД
- **down** – отмена действий, которые описаны в методе **up**

Изначально в методе **up** уже вызывается статический метод **create**, который создает таблицу **posts**.

```
Schema::create('posts', function (Blueprint $table) {
    $table->id();
    $table->timestamps();
});
```

Первый параметр – это название таблицы, а второй – анонимная функция с параметром **$table**, который описывает создаваемую таблицу.

Обратите внимание, что названия таблиц должны быть во множественном числе! Это правила фреймворка по умолчанию.

Внутри функции необходимо описать структуру создаваемой таблицы с помощью вызова методов структуры столбца. В таблице ниже перечислены некоторые из доступных методов.

Название метода	**Описание**
id	Создает поле первичный ключ с именем id
string	Создает строковое поле заданной длины
boolean	Создает поле типа boolean
timestamp	Создает поле с типом временной метки
timestamps	Создает две временные метки с именами created_at и updated_at. Поля содержат метку создания и метку обновления записи. Фреймворк сам следит за этими полями.

foreignId	Псевдоним метода unsignedBigInteger. Используется для создания внешнего ключа.

Более подробно про доступные методы можно прочитать в официальной документации https://laravel.com/docs/8.x/migrations#available-column-types

Примеры

```
$table->id(); // первичный ключ с именем id
$table->string('name', 100); // строка с именем name, длиной 100
$table->string('description'); // строка с именем description, длиной 255
$table->boolean('accepted');
```

Также к методам, которые описывают структуру столбцов можно применить методы модификаторы. В таблице ниже перечислены некоторые из модификаторов.

Название метода	**Описание**
default	Задает значение по умолчанию
nullable	Указывает, что поле может быть пустым при создании

Более подробно про доступные методы можно прочитать в официальной документации https://laravel.com/docs/8.x/migrations#column-modifiers

Примеры

```
$table->string('description')->nullable(); // Поле может быть пустым
$table->boolean('accepted')->default(false); // Если значение поля не было передано при создании, то значение будет false
```

Альтернативный способ создания миграции

Миграцию можно создавать автоматически при создании Модели. Про модели мы поговорим позже, но они в любом случае нужны если вы планируете работать с данными из этой таблицы.

Создать модель и миграцию к ней можно следующей командой

```
$ php artisan make:model Post -m
```

Данная команда создаст модель с именем **Post** в папке **app/Models**, а наличие флага **-m** приведет к созданию миграции. Названия моделей должны быть в единственном числе. При этом в миграции название таблицы будет уже во множественном числе.

Связи

Практически всегда при проектировании БД существуют связи между таблицами. Рассмотрим следующий пример. У нас есть автор и посты автора. Миграция для создания автора будет следующая

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

В миграции выше мы создаем таблицу авторов с 4 столбцами:

- id
- full_name
- created_at – создается при вызове timestamps()
- updated_at – создается при вызове timestamps()

Миграция поста будет выглядеть так

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

Сейчас таблица постов никак не привязана к автору. Для того чтобы их связать нам необходимо добавить поле user_id в таблицу постов. Сделать это можно несколькими способами.

Полный способ

```
$table->unsignedBigInteger('user_id');
$table->foreign('user_id')
      ->references('id')
      ->on('users')
      ->onDelete('cascade');
```

Сначала мы создаем поле **user_id** с помощью метода **unsignedBigInteger** и получаем **большое неотрицательное целое число**.

Дальше с помощью **foreign** мы указываем, что поле **user_id** это внешний ключ.
С помощью **references** указываем с каким ключом в другой таблице будет связь.
Метод **on** указывает на таблицу с которой происходит связь.
Метод **onDelete** указывает действие при удалении.

Краткий способ

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

Метод **foreignId** создает поле **unsignedBigInteger** с названием **user_id** и назначает его как внешний ключ, а **constrained** указывает на таблицу, с которой происходит связь. По умолчанию связь происходит со столбцом **id**, но это можно изменить во втором аргументе функции.

Работа с миграциями

Выполнение миграции

Для того чтобы применить миграцию необходимо выполнить следующую команду

```
$ php artisan migrate
```

Эта команда применяет только те миграции, которые ранее не были выполнены. Ранее выполненные миграции перечислены в таблице **migrations –** эта таблица создается при выполнении первой миграции.

Откат миграции

Отменить миграцию можно следующей командой

```
$ php artisan migrate:rollback
```

Перезапись всех миграций

В процессе разработки могут быть случаи, когда нужно изменить структуру миграции, но никаких важных данных в базе еще не содержится.

В таком случае можно воспользоваться командой fresh

```
$ php artisan migrate:fresh
```

Данная команда удаляет все таблицы из БД и применяет миграции заново.

Также есть команда refresh, которая работает немного по другому.

```
$ php artisan migrate:refresh
```

В начале все миграции отменяются (начиная с последней), а потого применяются заново (с самой первой).