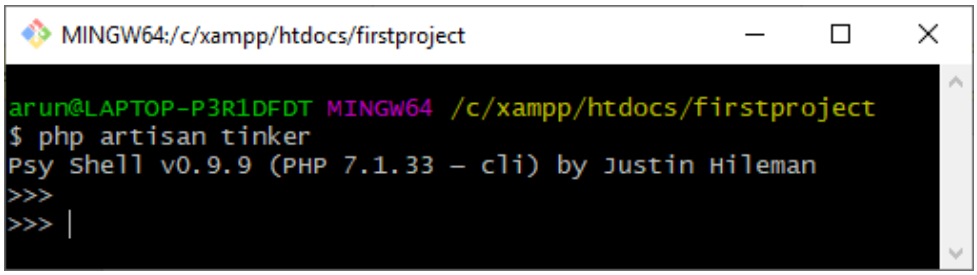
Laravel Tinker

Laravel Tinker allows you to interact with a database without creating the routes. Laravel tinker is used with a **php artisan** to create the objects or modify the data. The php artisan is a command-line interface that is available with a Laravel. Tinker is a command tool that works with a php artisan. A tinker plays around the database means that it allows you to create the objects, insert the data, etc.

- To enter the Tinker environment, run the command given below:

php artisan tinker



The above screen shows that the tinker environment has been created.

Creating data

- We can create the records in database tables by using the command-line tool. We use the following statement in the command-line tool that inserts the data directly in the database table:
  $post=App\Post::create(['title'=>'Akshay','body'=>'akshay is a software developer']);

```
MINGW64:/c/xampp/htdocs/firstproject

arun@LAPTOP-P3R1DFDT MINGW64 /c/xampp/htdocs/firstproject
$ php artisan tinker
Psy Shell v0.9.9 (PHP 7.1.33 – cli) by Justin Hileman
>>>
>>> $post=App\Post::create(['title'=>'Akshay','body'=>'akshay is a softw
=> App\Post {#3015
>>> $post=App\Post::create(['title'=>'Akshay','body'=>'akshay
 is a software developer']);
=> App\Post {#3015
     title: "Akshay",
     body: "akshay is a software developer",
     updated_at: "2019-11-26 12:57:58",
     created_at: "2019-11-26 12:57:58",
     id: 2,
   }
>>>
```

Output

When we execute the above statement, the data gets inserted in a posts table. We can view the inserted data in a **phpMyAdmin** shown in the below screenshot:

+ Options

		id	title	body	created_at	updated_at	deleted_at
Edit Copy Delete		1	Charu	technical Content Writer	NULL	2019-11-26 04:58:13	NULL
Edit Copy Delete		2	Akshay	akshay is a software developer	2019-11-26 12:57:58	2019-11-26 12:57:58	NULL

We can also use another way to insert the data by creating an object.

- First, we create the object.

```
MINGW64:/c/xampp/htdocs/firstproject

arun@LAPTOP-P3R1DFDT MINGW64 /c/xampp/htdocs/firstproject
$ php artisan tinker
Psy Shell v0.9.9 (PHP 7.1.33 – cli) by Justin Hileman
>>> $post= new App\Post
=> App\Post {#3019}
>>>
```

In the above screen, the highlighted line is creating the object, and the name of the object is $post. The $post is the object of the **App\Post** class.

We can also see whether the **$post** object has been successfully created or not of a given class by specifying the name of the object in a command-line tool.

```
MINGW64:/c/xampp/htdocs/firstproject
arun@LAPTOP-P3R1DFDT MINGW64 /c/xampp/htdocs/firstproject
$ php artisan tinker
Psy Shell v0.9.9 (PHP 7.1.33 — cli) by Justin Hileman
>>> $post= new App\Post
=> App\Post {#3019}
>>> $post
=> App\Post {#3019}
>>> |
```

The above-highlighted area shows that the $post object has been successfully created as **$post** shows the name of the class, **App\Post**.

- After the creation of an object, we will insert the data with the help of an object.

```
MINGW64:/c/xampp/htdocs/firstproject
arun@LAPTOP-P3R1DFDT MINGW64 /c/xampp/htdocs/firstproject
$ php artisan tinker
Psy Shell v0.9.9 (PHP 7.1.33 — cli) by Justin Hileman
>>> $post= new App\Post
=> App\Post {#3019}
>>> $post
=> App\Post {#3019}
>>> $post->title="Hema"
=> "Hema"
>>> $post->body="Hema is a hr"
=> "Hema is a hr"
>>> |
```

In the above screen, we have assigned the values to the column's title and body in a posts table by using the statements **$post->title and $post->body**, respectively. But, still, the data is not inserted in a table. To insert the data in the table, we need to use the statement given below:

$post->save(); // It saves the records in a database table.

When we type the $post in a command-line tool,

```
MINGW64:/c/xampp/htdocs/firstproject
arun@LAPTOP-P3R1DFDT MINGW64 /c/xampp/htdocs/firstproject
$ php artisan tinker
Psy Shell v0.9.9 (PHP 7.1.33 – cli) by Justin Hileman
>>> $post= new App\Post
=> App\Post {#3019}
>>> $post
=> App\Post {#3019}
>>> $post->title="Hema"
=> "Hema"
>>> $post->body="Hema is a hr"
=> "Hema is a hr"
>>> $post->save()
=> true
>>> $post
=> App\Post {#3019
     title: "Hema",
     body: "Hema is a hr",
     updated_at: "2019-11-27 06:33:17",
     created_at: "2019-11-27 06:33:17",
     id: 3,
   }
>>> |
```

The above-highlighted area shows that the record is saved in a posts table

Let's look at the **posts** table in phpMyAdmin

+ Options

	id	title	body	created_at	updated_at	deleted_at
Edit Copy Delete	1	Charu	technical Content Writer	NULL	2019-11-26 04:58:13	NULL
Edit Copy Delete	2	Akshay	akshay is a software developer	2019-11-26 12:57:58	2019-11-26 12:57:58	NULL
Edit Copy Delete	3	Hema	Hema is a hr	2019-11-27 06:33:17	2019-11-27 06:33:17	NULL

Finding Record

We can retrieve the records from the database in three ways:

- The first way is to use the **find()** method.

```
MINGW64:/c/xampp/htdocs/firstproject
arun@LAPTOP-P3R1DFDT MINGW64 /c/xampp/htdocs/firstproject
$ php artisan tinker
Psy Shell v0.9.9 (PHP 7.1.33 – cli) by Justin Hileman
>>> $post=App\Post::find(3)
=> App\Post {#3040
     id: 3,
     title: "Hema",
     body: "Hema is a hr",
     created_at: "2019-11-27 06:33:17",
     updated_at: "2019-11-27 06:33:17",
     deleted_at: null,
   }
>>> |
```

- The second way is to use the constraint, i.e., **where** clause.

```
>>> $post=App\Post::where('id',1)->first()
=> App\Post {#3041
     id: 1,
     title: "Charu",
     body: "technical Content Writer",
     created_at: null,
     updated_at: "2019-11-26 04:58:13",
     deleted_at: null,
   }
>>>
```

In the above screen, we are retrieving the record, which is having an '**id**' equal to 1. In this case, we use the first() method as the **first()** method is used to retrieve the single record.

```
MINGW64:/c/xampp/htdocs/firstproject
>>> $post=App\Post::where('id','>',1)->get()
=> Illuminate\Database\Eloquent\Collection {#3030
     all: [
       App\Post {#3042
         id: 2,
         title: "Akshay",
         body: "akshay is a software developer",
         created_at: "2019-11-26 12:57:58",
         updated_at: "2019-11-26 12:57:58",
         deleted_at: null,
       },
       App\Post {#3043
         id: 3,
         title: "Hema",
         body: "Hema is a hr",
         created_at: "2019-11-27 06:33:17",
         updated_at: "2019-11-27 06:33:17",
         deleted_at: null,
       },
     ],
   }
>>>
```

In the above screen, we are retrieving the records having an id greater than 1. In this case, more than one record is fetched, so we use the **get()** method. As get() method is used when an array of records is retrieved.

- The third way is to use **whereId()**.

```
   }
>>> $post=App\Post::whereId(2)->first()
=> App\Post {#3028
     id: 2,
     title: "Akshay",
     body: "akshay is a software developer",
     created_at: "2019-11-26 12:57:58",
     updated_at: "2019-11-26 12:57:58",
     deleted_at: null,
   }
>>>
```

Updating data

In this section, we will learn how to update the data in a database.

Let's understand through an example.

- First, we find out the object which we want to update.

```
MINGW64:/c/xampp/htdocs/firstproject

arun@LAPTOP-P3R1DFDT MINGW64 /c/xampp/htdocs/firstproject
$ php artisan tinker
Psy Shell v0.9.9 (PHP 7.1.33 - cli) by Justin Hileman
>>> $post=App\Post::find(2)
=> App\Post {#3040
     id: 2,
     title: "Akshay",
     body: "akshay is a software developer",
     created_at: "2019-11-26 12:57:58",
     updated_at: "2019-11-26 12:57:58",
     deleted_at: null,
   }
>>> |
```

In the above screen, we have retrieved the second record and stored in a **$post** object.

- Now we update the values of two columns, title, and body.

```
>>> $post->title="Prachi"
=> "Prachi"
>>> $post->body="business analyst"
=> "business analyst"
>>>
```

- To save the record in a database, we use the **save()** method.

```
>>> $post->save()
=> true
>>> |
```

In the above screen, the **save()** method returns true, which means that the record has been successfully updated in a database.

+ Options

←T→	id	title	body	created_at	updated_at	deleted_at
Edit Copy Delete	1	Charu	technical Content Writer	NULL	2019-11-26 04:58:13	NULL
Edit Copy Delete	2	Prachi	business analyst	2019-11-26 12:57:58	2019-11-27 08:51:04	NULL
Edit Copy Delete	3	Hema	Hema is a hr	2019-11-27 06:33:17	2019-11-27 06:33:17	NULL

Deleting data

Now, we will see how to delete the data from the database table.

Let's understand through an example.

- As we know that $post is an object that contains the second record, first, we apply the **delete()** on the **$post** object.

```
>>> $post->delete()
=> true
>>> |
```

The above screen shows that the **delete()** method returns true value, which means that the record has been deleted.

- Now, we will look at the database whether the record having an id equal to 2 is actually deleted or not.

Options

	id	title	body	created_at	updated_at	deleted_at
Edit Copy Delete	1	Charu	technical Content Writer	NULL	2019-11-26 04:58:13	NULL
Edit Copy Delete	2	Prachi	business analyst	2019-11-26 12:57:58	2019-11-27 09:00:22	2019-11-27 09:00:22
Edit Copy Delete	3	Hema	Hema is a hr	2019-11-27 06:33:17	2019-11-27 06:33:17	NULL

As we can observe from the above screenshot that the record of 'id' 2 is still available in the table, but a date is not null in the **deleted_at** column, which means that this record is soft-deleted.

- To delete the record permanently,

```
>>> $post->onlyTrashed()
=> Illuminate\Database\Eloquent\Builder {#3029}
>>> |
```

In the above screen, we use **$post->onlyTrashed()** means that the $post object contains only a trashed record.

To delete the trashed record permanently, we use the **forceDelete()** method on a **$post** object.

```
>>> $post->forceDelete()
=> true
>>>
```

The above screen shows that the **forceDelete()** method returns a **true** value, which means that the record is successfully removed from the table.

Let's look at the database:

Options

	id	title	body	created_at	updated_at	deleted_at
Edit Copy Delete	1	Charu	technical Content Writer	NULL	2019-11-26 04:58:13	NULL
Edit Copy Delete	3	Hema	Hema is a hr	2019-11-27 06:33:17	2019-11-27 06:33:17	NULL

In the above screenshot, we observe that the record of '**id**' 2 is deleted from the posts table.

Relationship with tinker

Till now, we find the relationship by using routes. Now, we will see the relationship through the tinker. In a laravel relationship topic, we read a one-to-one relationship with the help of the routes in which we find the post belonging to every user. Now, we find the post of every user in the tinker environment.

Let's understand through an example.

- First, we look at the data available in both the tables, users and posts table.

users table

+ Options

	id	name	email	email_verified_at	password	remember_token	created_at	upda
Edit Copy Delete	1	Charu	charu12@gmail.com	NULL	3456	NULL	NULL	NULL
Edit Copy Delete	2	Hema	hema67@gmail.com	NULL		NULL	NULL	NULL

posts table

+ Options

	id	user_id	title	body	created_at	updated_at	deleted_at
Edit Copy Delete	1	1	Software tester	Charu is a software tester	NULL	NULL	NULL
Edit Copy Delete	2	2	teacher	hema is a teacher.	NULL	NULL	NULL

- In this step, we will find the user.

```
MINGW64:/c/xampp/htdocs/firstproject

arun@LAPTOP-P3R1DFDT MINGW64 /c/xampp/htdocs/firstproject
$ php artisan tinker
Psy Shell v0.9.9 (PHP 7.1.33 - cli) by Justin Hileman
>>> $user=App\User::find(2)
=> App\User {#3026
     id: 2,
     name: "Hema",
     email: "hema67@gmail.com",
     email_verified_at: null,
     created_at: null,
     updated_at: null,
     country_id: 0,
   }
>>> |
```

In the above screenshot, we observe that the $user object contains the second user, i.e., a record of 'id' equal to 2 in the '**users**' table.

- Now, we implement the posts() method available in the User model through the **$user** object. The statement '**$user->posts**' calls the posts method of a User class.

```
MINGW64:/c/xampp/htdocs/firstproject
     country_id: 0,
   }
>>> $user->posts
=> Illuminate\Database\Eloquent\Collection {#3041
     all: [
       App\Post {#3042
         id: 2,
         user_id: 2,
         title: "teacher",
         body: "hema is a teacher.",
         created_at: null,
         updated_at: null,
         deleted_at: null,
       },
     ],
   }
>>> |
```

The above screen shows that the statement '**$user->posts**' retrieves the post of the user from the '**posts**' table.

← Prev Next →

For Videos Join Our Youtube Channel: Join Now

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

Learn Latest Tutorials

Splunk
SPSS
Swagger
Transact-SQL
Tumblr
ReactJS
Regex
Reinforcement Learning

R Programming

RxJS

React Native

Python Design Patterns

Python Pillow

Python Turtle

Keras

Preparation

Aptitude

Reasoning

Verbal Ability

Interview Questions

Company Questions

Trending Technologies

Artificial Intelligence

Artificial Intelligence

AWS Tutorial

AWS

Selenium tutorial

Selenium

Cloud Computing

Cloud Computing

Hadoop tutorial

Hadoop

ReactJS Tutorial

ReactJS

Data Science Tutorial

Data Science

Angular 7 Tutorial

Angular 7

Blockchain Tutorial

Blockchain

Git Tutorial

Git

Machine Learning Tutorial

Machine Learning

DevOps Tutorial

DevOps

B.Tech / MCA

DBMS tutorial
DBMS

Data Structures tutorial
Data Structures

DAA tutorial
DAA

Operating System
Operating System

Computer Network tutorial
Computer Network

Compiler Design tutorial
Compiler Design

Computer Organization and Architecture
Computer Organization

Discrete Mathematics Tutorial
Discrete Mathematics

Ethical Hacking
Ethical Hacking

Computer Graphics Tutorial
Computer Graphics

Software Engineering
Software Engineering

html tutorial
Web Technology

Cyber Security tutorial
Cyber Security

Automata Tutorial
Automata

C Language tutorial
C Programming

C++ tutorial
C++

Java tutorial
Java

.Net Framework tutorial
.Net

Python tutorial
Python

List of Programs
Programs

Control Systems tutorial
Control System

Data Mining Tutorial
Data Mining

Data Warehouse Tutorial
Data Warehouse