|  |  |
| --- | --- |
| Application | Code Used |
| - 1  - 2 | -1  TO INSTALL LARAVEL  Download php composer follow steps , then in Gitbash terminal type: |
|  |  |
| ---------------------------------------------------------------------------------------🡪  In views Folder |  |
| MVC  From 1)  To 2) | Routes.php calls function to controller,then from controller a function is called to the view. In views folder.  IMPORTANT NOTE, Use App\Model, i.e. App\User, in order for functions in model to be called.  An example of code responsible is shown below.      In routes.php   1. in PostController.php   Contact.blade.php |
| Parsing data to functions | 🡨-------- the code that allows this transition  1)  in routes.php  2)  in PostController.php  As seen variable id is passed through  3)  🡨--- html displayed |
| Parsing multiple variables | The code that allows this to occur      In routes.php 2 is id, 3 is name and 2 is password   1. in PostController.php   In the resources/views directory, post.blade.php |
| Parsing multiple variables, by using the blade templating engine | The code that allows this to occur.    🡨------------ to the left, is the template app.blade.php. @yield includes the division tags and @footer contains the body and html tags.  On post.blade.php ^ above, @section takes the division and @stop ends the division. @section(‘footer’) uses everything underneath  🡨------- and ends everything with body tag and html tag.  QUICKER WAY OF WRITING HTML |
| Reading multiple variables | 1. in routes.php 2. in PostController.php 3. in contact.blade.php. This code allows the names to be displayed on the contact page as shown to the left. |
| Add tables into the backend database | Code required    The .env file needs to be configured    Migration files    Command to make tables appear, as shown to the left |
| Add custom tables to the backend, Phpmyadmin | Code used    The migration file is created    As can be seen the variables properties are set. This code creates the table, the “blueprint”  Then the final command is called to allow for the tables to appear on the left |
| Edit tables in backend    To | Code used      Code is changed from above to down below      Final command, to allow the alterations to the tables, as shown to the left |
| Add column to tables in backend    To | Code used      Code above allows the variable is\_admin to the table    The code above is the final command to allow the table to appear on the left |
|  | Code used, for the table to the left |
| Make tables disappear than reappear in backend    Then | Code used |
| Insert data into tables in the backend    then | Code need for this    Fig 1  This is in routes.php  Make sure database connection is correct. As shown below. .env file    Use this url to allow the tables to appear. Fig 1 is the code for inserting data. SQL commands. |
| Insert data into tables in the backend, alternative method | Code that is used    The code above in routes.php    Url is used to insert data, allows code above to be executed |
| Insert data into tables in the backend, alternative method | Code that is used.    Fig 1  Fig 2  In order for this method to work, the variable, i.e. title and content, that will be inserted to the backend table, needs to be set up, as shown in fig 2.  Afterwards Fig3 is applied, ie URL allows the execution of code, that is Fig 1.  Fig 1 code is located in routes.php  Fig 2 is Post controller.  Fig 3 |
| Read data from backend data and put to front end    Fig 1  From backend table    As seen the title variable is put to the front end as shown on Fig 1 | Code used    Other method used, is shown below    Post::all() used the model Post. Note all models extends the model class. As shown below.    The class Post calls the table Posts, so the plural table is called when the class name is singular, unless use protected, as shown above. |
| Update data in table in backend    To | Code used    In routes.php. Update command is used |
| Update data in table in backend, alternative way    To | Code used    In routes.php    The post controller is called. Method is used from the model class.  Ie Post::find(2);    Url allows the above code to be executed |
| Update data in table in backend, alternative way    To | Code used  code in routes.php  Controller used is  Post controller is used. Post controller is needed for Post:where(id,2) to work    URL executes the code above in routes.php |
| Delete data from table in backend    To | Code used    In routes.php. Delete command is used    Url allows data to be deleted |
| Delete data from table in backend    To | Code used  In Fig 1, the code located in routes.php is used to delete data.  The post controller is needed. Fig 2  Fig 3, the url allows the code, in fig 1 to be executed.  Therefore overall deleting data in the tables. As shown to the left.    Fig 1    Fig 2    Fig 3 |
| Delete data from table in backend    To | Code used  in routes.php  The Post Controller model is needed, as show in the row above.    The url above allows the code above to be executed, therefore allowing data to be deleted, as shown on the left. |
| Use Soft Delete (Like recycling bin in windows, not deleting data permanently, but marked in backend table as deleted)    To | Code used  in routes.php  As can be seen to the left soft deletes is used in the Post Class. So Post::find()->delete() will allows soft deletes to work. $dates allows date to be added in deleted\_at column.  Look at migration below. Use code for migration. Php artisan migrate  Migration file:    URL cms.dev/softdelete allows the code in routes.php to be executed. |
| Read Soft Deletes (Like recycling bin in windows, not deleting data permanently, but marked in backend table as deleted)    Allows softdeletes and normal records to be read, as shown below | Code Used  in routes.php  The url cms.dev/readsoftdelete allows the above code to be executed. |
| Restore Soft Deletes (Like recycling bin in windows, restoring data from the recycling bin to the files in windows)    To | Code used  in routes.php  The url cms.dev/restore allows the above code to be executed. |
| Forced Delete, delete soft deletes (Like deleting files permanently from recycling bin in windows)    To | Code used  in routes.php  Alernative code to delete soft deletes is:    The url cms.dev/forcedelete allows the above code to be executed. All records including softdeletes are deleted. No timestamps, as was in soft deletes. |
| Find certain records in the backend database and put that record, certain values, to the front end | Code used    This code uses the model Post    Variable from title is displayed. The class Post calls the table Posts, so the plural table is called when the class name is singular, unless use protected, as shown above. |
| Join foreign tables together and pull out desired data. (One to One relationship)    Fig 1 (Users Table)    Fig 2 (Posts Table)    Fig 3  For certain attributes, the result shown below | Code Used  in routes.php  The model User is used.    To pull certain attributes:    As can be seen the table for users and posts, Posts, fig2 has the foreign key for posts, which is the primary key for users. Therefore $this->hasOne(); allows values to be returned, in which the foreign and primary key match. One to One match, One Post matches with one Users. |
| One to Many Relationship. This means that one user can have many posts, as an example    Backend Users Table    Backend Posts Table    Result displayed, primary key for user table is the same as user\_id for Posts table, hence the results are as there are. Shown above. Title 1 and Title 2 | Code Used    Model User is used, while calling the Posts method. Code shown below    The codes for both images, allows the results to be displayed on the left. |
| Many to Many relationship, pivot table, look at the example as follows    The backend table is as follows    Users Table    Role\_user table    Roles table  In summary the user\_id 1 is taken, then the value of role\_id is looked upon. The value in the primary key for the role table, hence the correct values is printed. Therefore the role of user one is printed out. | Code Used    Model User is used. |
| Many to Many relationship, with pivot another example. This example is basically a methodology on foreign keys.  All backend tables    Users Table    Role\_User Table    Getting one result from a table relative to another, using foreign keys | Code Used  In routes.php    This method in the model User, allows the role\_user table to be searched as using “with pivot”. So table User and table role is used, then “withPivot” allows the two table name to join, hence the role\_user table is used.  Therefore the value of created\_at for user id 1 is called, as shown to the result on the left. |
| Another example of many through relationships, getting one results from a table relative to another (Backend), using foreign keys    Countries Table    Users Table    Posts Table    Result that occurred from the code | Code Used  in routes.php  code in Country Model  One note, Users table is used first then posts table, therefore read from right to left. If errors occur, swap the table names around, so ‘App\User’ and then ‘App\Post’.  The primary id from countries table is taken, in this case one. After this the method “posts” is called. This means the users table is looked up and matches the primary id of countries to Users table. So “country\_id” in Users table. Then the primary id of users is matched to the posts table. Hence “user\_id” in posts table. Here the correct title attribute is received to the front end, as shown to the left. |
| Polymorphic relationship, Getting properties from One object relative to another. An example is all photos to one User. All Backend tables(beneath)    Photo Table    Posts Table    Users Table    The results as shown, all photos for one particular user. | Code Used  in routes.php    Function in the model Post. Photos function is called.  Afterwards the function imageable is called upon.    For this method to work, the method name, method\_id and method type in the table need to be the same. .ie. imageable, imageable\_id, imageable\_type  This method works by getting the primary id for posts, then looking up the photos table, where imageable\_id is the foreign key and imageable\_type being “App\Post” for the primary key for posts.  “\_id” and “\_type” are important for this method to work. |
| Polymorphic relation many to many – retrieving. Getting properties from One object relative to another. This is getting tags from taggables table. Backend tables, below.    Posts table    Taggables table    Tags Table    Displayed message for tags | Code Used  in routes.php  this code is Post Model  The code allows the primary key in Posts table to be received then looked upon the taggables table. Here if the taggable\_type and taggable\_id matches then the tag\_id is read. After the primary id of the tags table is used and the variable title is read.  The table name, table\_id and table\_type as well as method must remain the same. .i.e. taggables table, taggable\_id and taggable\_type. Therefore taggable\_id “1” and taggable\_type ”App\Post” is the primary id for posts table. Then continues onwards as described above. Tag\_id is the foreign key for Tags table. |
| Polymorphic relation many to many retrieving owner. Getting properties from One object relative to another. This is getting posts from tags table, using taggables as the pivot table. Backend tables, below.    Tags Table    Taggables Table    Posts Table    Results displayed from the code. | Code Used  in routes.php  in Tag model  Primary id ‘2’ is taken from the tags table, then used as the foreign key in the tags table. ‘tag\_id’. Then the ‘taggable\_id’ and ‘taggable\_type’ is read. In this case the id is 1 and type was App\Post. Therefore the primary id in post is read, that is primary id one. Then the title of that row is read and put to the front end. In this case ‘Title 1’. As stated before the method, table name, method\_id and method\_type need to be the same. Same format. See above rows if still confused. Look at the left hand column for pointers, hopefully can get the right code. |
| Creating data with tinker.    Posts table, inserting data. | Code Used    Command used in tinker command line is $post = App\Post::create(['title'=>'PHP post from tinker','content'=>'Php content from tinker']); |
| Inserting data with tinker. | Command Used  >>> $post->title = "New Title from this object"  >>> $post->content = "Yeah baby I'm coding and doing awesome"  >>> $post->save()  Result: type >>>$post, results shown |
| Read data with tinker | Command Used  >>> $post = App\Post::find(4); |
| Soft Deletes, then hard deletes with tinker    To | Command Used  >>> $post = App\Post::find(4);  >>> $post->delete() (Soft Delete)  >>> $post->forceDelete() (Hard Delete) |
| Note: Tinker is basically like using routes | Command Used  >>> $user->roles |
| CREATE SHORTHAND NAME FOR WEB BASED APPLICATION NAME    Figure 1    Figure 2 | File Path Used  Code need to make this work.  Figure 1: C:\xampp\apache\conf\extra\httpd-vhosts.conf  Figure 2: C:\Windows\System32\drivers\etc\hosts |
| SET DATABASE CONNECTION UP    Backend Database | Connection file used  .env file |
| One To One relationship, CRUD. (Inserting One address relative to one user)  Backend Table    Addresses Table    Users Table    Insert Command | Code Used  in routes.php  in Users Model  in Address Model. Allows ‘name’ field in addresses table to be entered in .i.e. data inserted in this field.  This code, ‘$user->address()->save($address);’ allows user id, the primary id in users table to be put as the foreign key in addresses table. So user\_id 1 in ‘addresses’ table is primary id 1 in ‘Users’ table. |
| One to One relationship, CRUD. (Update One address relative to One User)  Backend Tables    Addresses Table  TO      Insert Command | Code Used  in routes.php  This code finds the userId that matches, in the address table then updates the address.  “$address = Address::whereUserId(1)” looks for ‘User\_id’ in the addresses table, in this case ‘1’. |
| One to One relationship, CRUD. (Read One address relative to One User)  Backend Table    Addresses Table    URL typed. | Code Used  in routes.php  User Model Used.  The data that has ‘User\_id’ 1 is read from the backend table, as shown on the left. Scroll up if still not sure. |
| One to One relationship, CRUD (Delete One address relative to One User)  Backend Table    TO      URL Typed | Code Used  in routes.php  User Model Used.  The data that has ‘User\_id’ 1 is deleted from the backend table, as shown on the left. Scroll up if still not sure. |
| One to Many relationship, CRUD (Update One Posts relative to one user)  Backend Table    Posts Table  To      URL Used | Code Used    in User Model  As can be seen in the code, the ‘User\_id’ 1 is located in the posts table, then the data is updated. As shown on the left, look clearly. |
| One to Many relationship, CRUD (Delete One Posts relative to One user)  Backend Table    Posts Table  To      Users Table    URL Used | Code Used  in routes.php  in User Model  Data is received from the Posts table where User\_id is 1. From therefore, the primary id in posts table that is one is deleted. User\_id is the foreign key for the primary id in Users tbale. Therefore the results are shown on the left. If confused look up foreign key and primary key relationship. |
| One to Many relationship, Another Example, CRUD (Delete One Posts relative to One user)  Backend Table    To    Posts Table    Users Table    URL Used | Code Used  in routes.php  in Users Model  All data in the posts table are deleted with User\_id 1. User\_id is the foreign key for the primary id of users. Therefore in the example, shown to the left, all posts relative to user “Dinesh Mittal” are deleted. |
| Many to Many relationship. CRUD. (Inserting One role relative to One user)  Backend Table    Users Table    Role\_user table    Roles Table    URL Used | Code Used  in routes.php  in User Model  The code allows the primary id 1 in the users table to be taken and inserted as ‘User\_id’ in the pivot table. ‘Role\_user’ table. Then in ‘role\_id’ in role\_user table, an id is inserted which auto increaments. After, the roles table is used to insert the role. As shown in the roles table. Therefore this convention allows a user to have many roles. As more roles are added the role\_id is role\_users table is increased, whilst user\_id can remain the same. .i.e. role\_id 1,2,3…. . (Look carefully at example). Still not sure, go to Section 15, lecture 98. Edwin Diaz Laravel. Udemy.  Pivot table name convention is as follows. $this->belongsToMany(‘App\X’). Y is Users table.  Therefore the name convention is X\_Y |
| Many To Many relationship. CRUD (Read One role relative to One User)  Backend Table    Users Table    Role\_user table    Roles Table    URL Used | Code Used  in routes.php  in User Model  The code allows the primary id of 1 in the users table to be used. After, the method ‘$this->belongsToMany(‘App\Role’)’ allows the table role\_user to be used. This table is used because Role is ‘App\Role’ is taken and the code ‘$user = User::findOrFail(1), $user->roles’ allows the name user, since the model user is used. Hence the name of the table ‘role\_user’. As model name in the singular for the table name. Model user, table users. This table is used as a pivot, therefore the role\_id that matches the User\_id, in this case 1 is used. Role\_id is the foreign key for the primary id of Roles table. Therefore the attributes that are associated with that particular primary id are read from the database, ‘roles’ table and ‘put’ to the front end. As shown to the left. Dinesh’s role is Administrator. |
| Many To Many relationship. CRUD (Update roles relative to One User)  Backend Table    Users Table    Role\_user table    Roles Table  To      URL Used | Code Used  in routes.php  In User Model  The code allows the primary id of 1 in the users table to be used. After, the method ‘$this->belongsToMany(‘App\Role’)’ allows the table role\_user to be used. This table is used because Role is ‘App\Role’ is taken and the code ‘$user = User::findOrFail(1), $user->roles’ allows the name user, since the model user is used. Hence the name of the table ‘role\_user’. As model name in the singular for the table name. Model user, table users. This table is used as a pivot, therefore the role\_id that matches the User\_id, in this case 1 is used. Role\_id is the foreign key for the primary id of Roles table. Therefore the attributes that are associated with that particular primary id are updates from ‘Administator’ to ‘subscriber’. As shown on the left. |
| Many To Many relationship. CRUD (Delete roles relative to One User)  Backend Table    Users Table    Role\_user table    Roles table  To (Deleted, primary id key 3)      URL Used | Code Used  in routes.php  in User Model  The code allows the primary id of 1 in the users table to be used. After, the method ‘$this->belongsToMany(‘App\Role’)’ allows the table role\_user to be used. This table is used because Role is ‘App\Role’ is taken and the code ‘$user = User::findOrFail(1), $user->roles’ allows the name user, since the model user is used. Hence the name of the table ‘role\_user’. As model name in the singular for the table name. Model user, table users. This table is used as a pivot, therefore the role\_id that matches the User\_id, in this case 1 is used. Role\_id is the foreign key for the primary id of Roles table.  The code above ‘$role->whereId(3)’ allows the primary id, 3, in ‘roles’ table to be deleted. As shown to the left. |
| Many To Many relationship. CRUD (Attaching roles relative to Users)  Backend Table. Attaching one property to another property  Role\_User table  TO  Role\_User table    URL Used | Code Used  in routes.php  in User Model  The function ‘$this->belongsToMany()’ allows the pivot table ‘role\_user’ to be used. Therefore the code above allows the role id of two to be added to the ‘role\_user’ table with user id of one. Hence results shown on the left. |
| Many To Many relationship. CRUD(Detaching roles relative to Users)  Backend Table. Detaching one property to another property  Role\_user table  Role\_User table    URL Used | Code Used  in routes.php  in User Model  The function ‘$this->belongsToMany()’ allows the pivot table ‘role\_user’ to be used. Therefore the code above allows the role id of two to be removed to the ‘role\_user’ table with user id of one. Hence results shown on the left.  ‘$this->roles()->detach()’ allows all role\_id to be removed. So a blank ‘role\_user’ table will occur. |
| Many To Many relationship. CRUD (Synching roles relative to Users)  Backend Table. An metaphorical example is like itunes. If delete on one end everything gets deleted on other end. What every upload or removed gets synched.  Role\_User table  Role\_User table    URL Used | Code Used  in routes.php  in User Model  The code above allows role\_id 6 and 7 to be attached to the role\_user table, where user\_id is one. As well as being attached, role\_id 1 and 3 are removed, detached. .i.e. a substituion occurs. Look at example on left if still confused. |
| Eloquent Polymorphic Relationship CRUD Database. Inserting Data. Inserting photos relative to one staff.  Backend Tables  Staff(Plural) table    Photos table    URL Used | Code Used  in routes.php  code in Staff Model, function imageable is called  code in Photo Model  The code in routes.php allows the method imageable, in Photo Model to be called. ‘$staff = Staff::find()’ then ‘$staff->photos()’, this allows photo method in staff model to be called. After this the photo method calls imageable method in photo model. ‘$this->morphMany(‘App\Photo’,’imageable’)’ convention ‘‘$this->morphMany(‘App\Table Name/model’,’function in model’)’. Therefore the function imageable is called. The function allows the photos table to be accessed. Hence ‘$staff->photos()->create()’,in routes.php, allows data to be entered in the table. As shown to the left. |
| Eloquent Polymorphic Relationship CRUD Database. Updating Data. Updating photos relative to one staff.  Backend Tables  Staff(Plural) Table    TO    Photos table    URL Used | Code Used  in routes.php  code in Staff Model, function imageable is called  code in Photo Model  The code in routes.php allows the method imageable, in Photo Model to be called. ‘$staff = Staff::find()’ then ‘$staff->photos()’, this allows photo method in staff model to be called. After this the photo method calls imageable method in photo model. ‘$this->morphMany(‘App\Photo’,’imageable’)’ convention ‘‘$this->morphMany(‘App\Table Name/model’,’function in model’)’. Therefore the function imageable is called. The function allows the photos table to be accessed. Hence the code in routes.php allows the ‘path’ variable in photos table to be updated, as shown to the left. |
| Eloquent Polymorphic Relationship CRUD Database. Deleting Data. Updating photos relative to one staff.  Backend Tables  Staff(Plural) Table    Photos Table  To    Photos Table    URL Used | Code Used  in routes.php  code in Staff Model, function imageable is called  code in Photo Model  The code in routes.php allows the method imageable, in Photo Model to be called. ‘$staff = Staff::find()’ then ‘$staff->photos()’, this allows photo method in staff model to be called. After this the photo method calls imageable method in photo model. ‘$this->morphMany(‘App\Photo’,’imageable’)’ convention ‘‘$this->morphMany(‘App\Table Name/model’,’function in model’)’. Therefore the function imageable is called. The function allows the photos table to be accessed. Hence the code in routes.php allows data in photos table with primary id of 1 to be deleted. As shown to the left. |
| Eloquent Polymorphic Relationship CRUD Database. Assign values. Automatically, ie fill in blanks for photos by using variables from staff  Backend Tables    To    Photos table    URL Used | Code Used  in routes.php  code in Staff Model, function imageable is called  code in Photo Model  The code in routes.php allows the method imageable, in Photo Model to be called. ‘$staff = Staff::find()’ then ‘$staff->photos()’, this allows photo method in staff model to be called. After this the photo method calls imageable method in photo model. ‘$this->morphMany(‘App\Photo’,’imageable’)’ convention ‘‘$this->morphMany(‘App\Table Name/model’,’function in model’)’. Therefore the function imageable is called. The function allows the photos table to be accessed. Hence the code In routes.php allows the photos table’s variables to be filled, as shown to the left. |
| Eloquent Polymorphic Relationship CRUD Database. Un-Assign values. Automatically, i.e. Unfill in blanks for photos.  Backend Tables    TO      URL Used | Code Used  in routes.php  code in Staff Model, function imageable is called  code in Photo Model  The code in routes.php allows the method imageable, in Photo Model to be called. ‘$staff = Staff::find()’ then ‘$staff->photos()’, this allows photo method in staff model to be called. After this the photo method calls imageable method in photo model. ‘$this->morphMany(‘App\Photo’,’imageable’)’ convention ‘‘$this->morphMany(‘App\Table Name/model’,’function in model’)’. Therefore the function imageable is called. The function allows the photos table to be accessed. Hence the code In routes.php allows the photos table’s variables to be unfilled, as shown to the left. |
| Database-Eloquent Polymorphic Many to Many relationship CRUD.(Inserting Data). Inserting tags to videos and posts.  Backend Table (Data Inserted)  Posts Table  Taggables Table  Tags Table  Videos Table    URL Used | Code Used  in routes.php  in Post Model and Video Model  The code allows the primary id of tags, by allowing the variable ‘$tag1 = Tag::find(1); , ‘$post->tags()’, this allows the taggable type to be received. In this case ‘App\Post’. The convention is ‘$X->tags()’. For taggable type ‘App\X’. The code allows the values for tag\_id, taggable\_id which is the primary id for posts or videos in this case, the code ‘$post = Post::create(); allows the primary id to be received, of posts table, and taggable\_type to be inserted to the taggables table. This happens because of the functions tags() and the code in routes.php ‘$post->tags()->save();’. Results shown to the left. Overall receiving all variables and putting to taggables table, which acts as a ‘lookup’ table. |
| Database-Eloquent Polymorphic Many to Many relationship CRUD.(Updating Data). Updating tags to videos and posts.  Backend Table (Data Updated)  Posts Table  Taggables Table  Tags Table  To    Videos Table    URL Used | Code Used  in routes.php  in Post Model and Video Model  The code above allows the primary id of posts to be received. ‘$post = Post::findOrFail() and $post->tags’. This code, allows the taggable type to be received. ‘$post->tags.’Hence ‘App\Post’. The function tags allows taggable table to be called. The variables are looked up, and the tag\_id is received. The variable will be the primary key for tags. From where the code ‘$tag->whereName() etc…’ allows the data to be updated, as shown to the left. |
| Database-Eloquent Polymorphic Many to Many relationship CRUD.(Updating Data). Updating tags to videos and posts.  Backend Table (Data Updated). SYNC    TO    Taggable Table | Code Used  in routes.php  in Post Model and Video Model  The code allows tag\_id to be updated to one and nothing else, where taggable\_id is one. Look above to previous examples if still confused. |
| Database-Eloquent Polymorphic Many to Many relationship CRUD.(Deleting Data). Deleting tags that matched posts.  Backend Table  Tags Table  TO  Tags Table  Taggables Table  Posts Table    URL used | Code Used  in routes.php  in Post Model and Video Model  The code above allows the primary id of posts to be received. ‘$post = Post::find() and $post->tags’. This code, allows the taggable type to be received. ‘$post->tags.’Hence ‘App\Post’. The function tags allows taggable table to be called. The variables are looked up, and the tag\_id is received. The variable will be the primary key for tags. From where the code ‘$tag->whereId(2)->delete etc…’ allows the data to be deleted, as shown to the left. Primary id of 2 in tags table is deleted. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |