**1 Steps to run the project**

Enter to the project directory

* modify the config/database.yml file check the database password
* run bundle install
* rake db: create ( if needed )
* rake db: migrate ( if needed )
* run server (rails s)
* open in browser ( localhost:3000)

**2**

Take a Back up the data of the database for the house project using **Export** method in **phpMyadmin** or some other method

We can use the **YAML**  library in ruby to take a backup of the entire database.

Open the rails console and get the entire data in a variable and then open a file with a text file and we can dump the data of the entire database in to a file and create the back up.

@h=House.all

f=File .new (“file path”, “mode”)

YAML.dump(@h,f)

f.close

**3 Association**

**Creating the association**

* Create another model for the person Table

Rails g model person name: string title: string phone: string

**Condition for Association**

For a single house we are having many members.

So House Model is related to Person by  **one-to-many**

Add in the lines in the respective model class and in migration file

* **house.rb**

has\_many: persons

* **person.rb**

belongs\_to: house

* **20120106121520\_create\_people.rb**

t.references :house

**4**

Run the migration to make the changes in the database tables.

Rake db: migrate

Open the rails console and check the association

rails c

@h=House.find(1)

@p=Person.new(:name=>”kiran”, :title=>”father”, : phone=>345543)

@h.persons<<@p

If this is working then association is made correctly.

**5 Controller**

Create a new controller to add the functionality to the person table such as

* Add a member to the table
* Delete a member from the table

Edit the person details (if any details changes like address changes)

Rails g controller person  **or**

Rails g controller person new show edit

(This line will add the respective view pages along with controller action and also add the following lines in the routes file

#get "person/index"

#get "person/new"

#get "person/show”)

**6 Routes**

Add the following lines in the routing file so that any request will get the correct response and person controller can be accessible.

* **Routes.rb**

Resources: houses do

Resources: person

End

* **Uncomment the last line in the routes.rb file**

match ': controller(/:action(/:id(.:format)))'

**7 Controller**

Code in the controller and view files

We need to Add, delete and update the member details

To perform the basic CRUD operation we need 6 action methods in the **person\_controller**

1. **Show action**

To display the members in a specied family we need this action

def show

@h=House.find(params[:id])

@per=@h. persons.all

End

The first line find the given id and get the details for a particular house and the second line will fetch all the member s for a that house by calling the association methods.

1. **New action and Create action**

This two action will create a new member for a particular house

def new

@h=House.find(params[:id])

@per=@h.persons.new

End

def create

@h = House.find(params[:id])

@per=@h.persons.create(params[:person])

if @per.save

flash[:notice]="person detail entered"

redirect\_to :action=>'show',:id=>@h.id

else

render :action =>'new'

end

end

the new method will get the house details for a particular id and create a new instance variable for the member for the particular house

the create method gets the id for the house and saves the form data for the member associated with the particular house.

1. **Edit action and Update action**

This two methods will change any particular details for any member

def edit

@per=Person.find(params[:id])

end

def update

@per=Person.find(params[:id])

@h=@per.house

if @per.update\_attributes(params[:person])

flash[:notice]="person detail updated"

redirect\_to :action =>'show',:id=>@h.id

else

render :action=>'edit'

end

end

the edit action will get the particular person details for the provided id from the database table the update method will get the form data from the edit view page and save it to the provided @per instance variable and save it to the database. And redirect or render the provided view page according to the condition.

1. **Detete Action**

This action will delete the particular member from database.

def delete

@per=Person.find(params[:id])

@h=@per.house

if @per.delete

flash[:notice]="person detail deleted"

redirect\_to :action=>'show',:id=>@h.id

end

end

this action get the person id from the index page and delete the member details by calling the method delete for the instance variable and redirect to the view page ‘show’ and display a flash message.

**8. Views**

**Show.html.erb**

@per variable in show method in person\_controller fetch all the members for a particular house .

In show.html.erb file the each method will fetch one member at a time from the @per array and display in the browser.

‘Link\_to’ helper method will provide the link to different action methods and view files.

**New.html.erb and Edit.html.erb**

this view page will create a form for the member table by using the form\_for tag helper the only difference in new and edit is both passes different id create take house id and update take person id .

there are some link to tag helper to provide link to other view pages.

**9. Adding Validation**

We can add validation in the model classes.

**House.rb**

Some respective validation is added to save only validated data in database. The dependent key word will delete the associated member data once a house is deleted.

**has\_many: persons,: dependent => :destroy**

**validates :houseno,:address1,:address2,:address3,:address4,:owner ,:presence => true**

**person.rb**

**belongs\_to :house**

validates :name,:title,:phone ,:presence => true

validates :phone , :numericality =>{ :only\_integer => true }

**Q:2**

array1 = [

{"type" => "Chipmunk", "actions" => ["Squeak", "Wear Baseball Caps"]},

{"type" => "Dog", "actions" => ["Woof", "Fetch"]},

{"type" => "Cow", "actions" => ["Moo", "Tip"]},

{"type" => "Horse", "actions" => ["Neigh", "Gallop"]},

{"type" => "Pig", "actions" => ["Grunt", "Eat"]},

]

bytes: 419

array2 = [

{:type => "Chipmunk", :actions => ["Squeak", "Wear Baseball Caps"]},

{:type => "Dog", :actions => ["Woof", "Fetch"]},

{:type => "Cow", :actions => ["Moo", "Tip"]},

{:type => "Horse", :actions => ["Neigh", "Gallop"]},

{:type => "Pig", :actions => ["Grunt", "Eat"]},

]

bytes: 332

In case of first array1 all “type” and “action ” are strings and in the second array2 all :type and :action are symbol .

A symbol in Ruby is an instance of the class Symbol. A symbol is defined by prefixing a colon with an identifier. **: type**, **:action**, **:id** etc. are examples of symbols.

Symbols generally have performance benefits.

Symbols provides the efficient use of memory and faster performances

Symbol is immutable

**:** **type** and  **:action** they share a particular memory they are stored as a integer in the program and the programmer can use symbol as a string.

Here in this case as symbols share a same memory location so they take much less memory then the string.

One example

In case of rails project every path is given by symbols.

The link\_to tag helper takes hash as a parameter and every time the link\_to is used then the :controller ,:action and :id will take same memory but if you take these as a string they share much memory and also in comparison they take more time. As string comparison is slow then integer comparison (: symbol).

So for performance we use symbol as for string but symbol have some disadvantage like

Symbols can't be changed at runtime

Symbols class doesn’t have the rich functionality like String class.