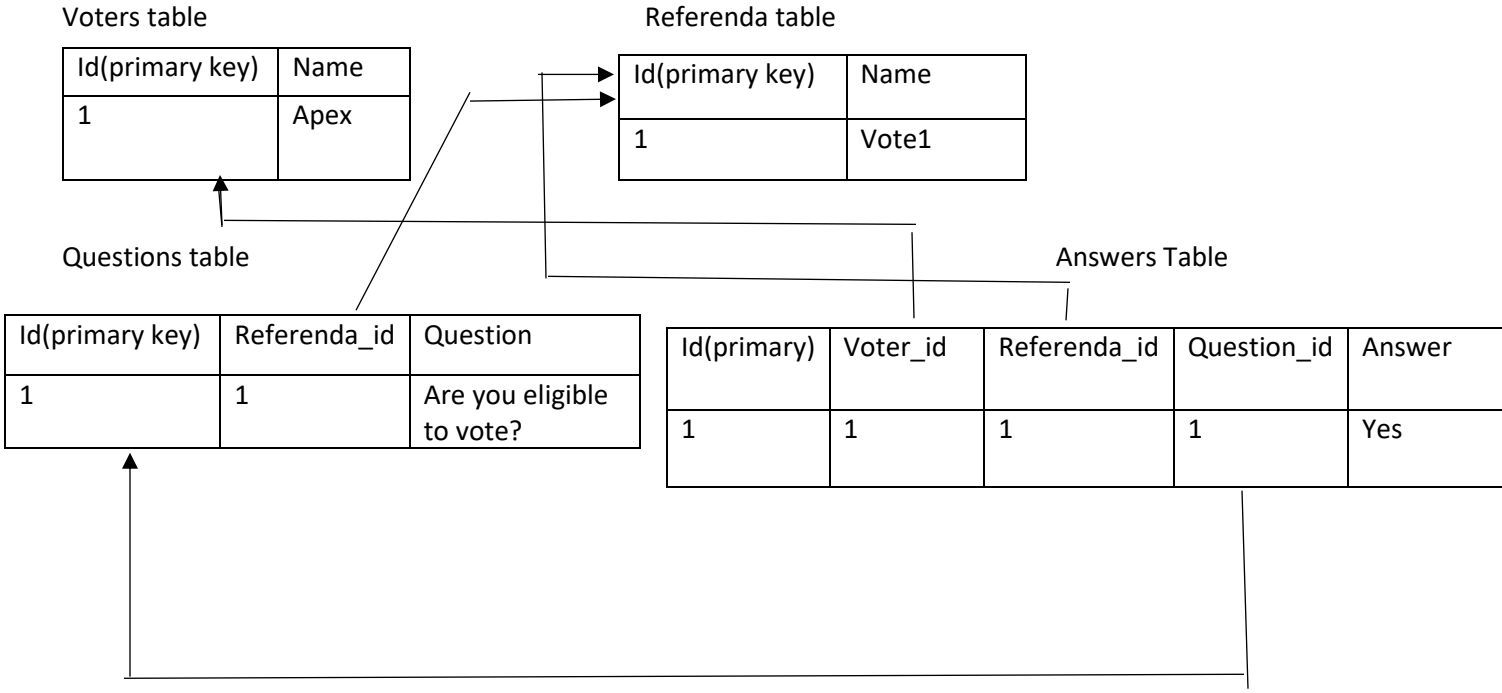
NAME: SHOBOWALE AYOMIDE ENIOLA

1). Entity relationship diagram



Normalized schema

```
CREATE TABLE Voters(
[ID] INT PRIMARY KEY IDENTITY,
[Name] VARCHAR(100)
)
GO
```

```
CREATE TABLE Referenda(
[ID] INT PRIMARY KEY IDENTITY,
[Name] VARCHAR(250)
)
GO
```

NAME: SHOBOWALE AYOMIDE ENIOLA

```
CREATE TABLE Questions(
[ID] INT PRIMARY KEY IDENTITY,
[Refereda_id] INT(100),
[Question] VARCHAR(250),
CONSTRAINT FK_Questions_Referenda FOREIGN KEY ([Referenda_id]) REFERENCES
[Referenda] ([Id])
)
GO
```

```
CREATE TABLE Answers(
[ID] INT PRIMARY KEY IDENTITY,
[Voter_id] INT(100),
[Question_id] INT(100),
[Answer] VARCHAR(250),
CONSTRAINT FK_Answers_Voters FOREIGN KEY ([Voter_id]) REFERENCES  [Voters]
([id]),
CONSTRAINT FK_Answers_Referenda FOREIGN KEY ([Referenda_id]) REFERENCES
[Referenda] ([id]),
CONSTRAINT FK_Answers_Questions FOREIGN KEY ([Question_id]) REFERENCES
[Questions] ([id])
)
GO
```

b). For the results of a given referendum question, without querying a single voter's voting record.

A simple count query on the answers table would suffice.

For Number Of Yes -> Select count(*) from Answers where Referenda_id = $id AND Answer = 'Yes'

For Number Of No -> Select count(*) from Answers where Referenda_id = $id AND Answer = 'No'

c). Privacy was considered in the design of the schema, which resulted in the adoption of all database normalization laws in its creation. The tables all have unique primary keys, all data within each table are also stored in individual columns in its most reduced form, and also there were no repeating groups in the tables.

NAME: SHOBOWALE AYOMIDE ENIOLA

2). To change source code comments In commit M before submitting, you can use

"`git commit –amend`" command.

a). On the command line, navigate to the repository that contains the commit you want to amend.

b). Type `git commit --amend` and press **Enter**.

c). In your text editor, edit the commit message, and save the commit.

ii) Also N' is not the same with N, N' is the difference between the old commit and N.

3). This process is called Dependency Injection. In Laravel, dependency injection is the process of injecting class dependencies into a class through a constructor or setter method. This allows your code to look clean and runs faster. It involves the use of a Laravel service container, a container used to manage class dependencies.

To implement this;

a). you first create the function in its parent class.

b).Then you require it in the class in which you want to inject its dependencies (use App\Repositories\class)

c).Then it can be used in any function in the current class

```
class MyController extends Controller
{
   private $provider;

   public function __construct(MyContentProvider $provider)
   {
      $this->provider = $provider;
   }

   public function getContent()
   {
      return view('index', [ 'content' => $this->provider->get() ]);
   }
}
```

NAME: SHOBOWALE AYOMIDE ENIOLA

4). The code written was not in-line with the accepted standard of Laravel array_map(). Also in newer versions of laravel toArray() should be replaced with all(), because in case of eloquent collection, toArray() will try convert your models to array too. All() will return an array of your model as it is.

The code can be refactored like;

```
class Post extends Model
   {
        Public function func($comment_id)
        {
          return Comment::find($comment_id)->get();
        }

        public function comments()
        {
            $comment_ids = array_map(function ($comment){
          return $comment->id; }, Comment::where("post_id", $this->id)->get()->all());

          $s_commentid = array_map(function (“func”), $comment_ids);
         }
   }
```

5). `The unix command`

“ grep -Zlir users /tmp | xargs -0 -r stat --format='%Y+%n' | sort -t+ -k 1,1nr | cut -d+ -f2-“

6). The !important rule in css is used to give priority and importance to a property/value. When used it overrides all previous styles for that specific property on that element.

Front-end developers advice that it be used with caution because it will affect other tags and elements that you may not want to target, then would have to hardcode to revert back to the original style.

Although it can be useful in some of it applications. For example where you have external features like graphs and charts that depends on external css for its styling and looks. If you want those attributes changed, then you would have to write css locally and then make those properties important, so as to override the ones coming from the external library.

NAME: SHOBOWALE AYOMIDE ENIOLA

7). Output:

{ value: 3, done: false }
{ value: 4, done: true }
{ value: undefined, done: true }

Generators are functions that can be exited and later re-entered. Their context (variable bindings) will be saved across re-entrances. When the iterator's `next()` method is called, the generator function's body is executed until the first `yield` expression, which specifies the value to be returned from the iterator or, with `yield*`, delegates to another generator function. The `next()` method returns an object with a `value` property containing the yielded value and a `done` property which indicates whether the generator has yielded its last value, as a boolean. Calling the `next()` method with an argument will resume the generator function execution, replacing the `yield` expression where an execution was paused with the argument from `next()`. A `return` statement in a generator, when executed, will make the generator finish (i.e. the `done` property of the object returned by it will be set to `true`). If a value is returned, it will be set as the `value` property of the object returned by the generator. Much like a `return` statement, an error thrown inside the generator will make the generator finished -- unless caught within the generator's body. When a generator is finished, subsequent `next()` calls will not execute any of that generator's code, they will just return an object of this form: `{value: undefined, done: true}`.

Code Explanation
In this piece of code, the first call of next executes and "3 < 4" therefore result is 3, it increases to 4 in the while loop and since it ran it the generator, it means it isn't done, so "done is set to false" at the first iteration. At second iteration, 4 is not < 4, so an error is thrown in the generator, which made it finish, therefore setting "done to true", 4 escapes the while loop and 4 is returned. Further iteration isn't allowed by the, as further tests fail at this point, setting the "value to undefined" and done to be true once again.