1. **What is a class?**
   * Class: Collection of Students + Chairs + Others ==> Grouping something
   * Family ==> Family Members + Others
     + Class ==> Properties + Methods
     + Class ==> Methods
     + Class ==> Properties
     + Class ==> Variables + functions
   * House ==> Class
   * Product
   * Register
   * Employee
     + adding
     + view
     + delete
     + update

class <class-name> {

properties

methods

}

1. **What is an object? | Static and Non-Static Members**
   * How can we access the Class members?
     + By creating an Object
   * What is an Object?
     + An Object is an Instance of a Class, it's like a blueprint of a Class
   * What is meant by instance of a class?
     + Allocating memory to the Class members is called instance of a Class
   * Creating Object

new operator, we can create an Object:

$variable = new <ClassName>();

* + Static members
    - The Class members which are created with Static keyword are called Static members
  + Non-Static members
    - The Class members which are not created with Static keyword are called Non-Static members
  + How can we access Static members of a Class?
    - Using Class name with Scope Resolution Operator (::)
  + Scope Resolution Operator (::)
    - By this operator, it's possible to access Static members of a Class, Constants of a Class.
    - <ClassName>::CONSTANT\_NAME
  + Static Variables => ClassName
  + Non-Static Variables => Object
  + Static Methods => ClassName or Object
  + Non-Static Methods => ClassName or Object

1. **$this keyword in PHP | self:: keyword**
   * $this
     + Always refers the current Class Object
     + Scope - inside of a class only
     + Using $this, you can only access non-static members of a class only

$this ==> non-static

* + self::
    - Scope - inside of a class onlyUsing self::, you can only access static members of a class only

self:: ==> only static

1. **Constructor and Destructor in PHP**

Constructor and Destructor

Default:

public function \_\_construct(){ connected to DB }

public function \_\_destruct(){ close DB }

The construct function gets called automatically at the time of Object creation.

* + The destruct function gets called automatically at the time of class execution completed.

1. **Examples using Constructor and Destructor**

=====================================================================================================================================================

6- Inheritance in PHP

Inheritance

Inheritance is the process of sharing the Base Class members into the Sub Classes

Inheritance is implemented by using "extends" keyword

Parent Class and Child Class

Base Class and Sub Class

=====================================================================================================================================================

7- Model View Controller MVC

MVC Architecture

MVC is a design pattern, which is used to organize the code into a structured manner

Presentation Logic = HTML CSS

Business Logic = PHP & MySQL

The main GOAL of MVC Architecture is sepaating the Presentation Logic with Business Logic

MVC - Model View Controller

Model:

A Model is a class file. A model always deals with database operations

Models:

- register.php

- login.php

View:

A View is also a file which contains Presentation Logic

Views:

- register.php

- login.php

Controller:

The Controller is THE HEART of MVC Architecture. In MVC, each and every request will be handled by the Controller only.

register.php

HTML

CSS

JS

PHP & MySQL

=====================================================================================================================================================

8- CodeIgniter 3 setup

$this keyword

inheritance

MVC Architecture

What is a framework?

A framework is created by using a programming language

CodeIgniter => PHP

Laravel => PHP

ZEND => PHP

CakePHP => PHP

PHP: by using this we can develop web applications

CodeIgniter: by using this we can develop web applications

validate email / mobile in php = 10 ~ 15 lines

validate email / mobile in framework = 1 line

CI3

CI4

Concise code => Reduce the number of ilnes of

XAMPP tool

htdocs

Cache

Config => Heart of CI3

Helpers

Libraries => Some predefined classes + our own classes

Models

Views

Controllers

=====================================================================================================================================================

9- CodeIgniter 3 URLs

Types of URLs:

Query string based URLs: http://example.com/test.php?id=1&name=ram

Segment based URLs: /segment/ http://example.com/test/1/ram

http://example.com/news/article/my\_article

example.com = domain

test / news = segment1 <-------- Controller

1 / article = segment2 <-------- Method or Function

ram / my\_article = segment3 <-------- Parameter to the methods

class News {

function article(my\_article) {

// some logic

}

function test() {

// some logic

}

}

=====================================================================================================================================================

10- Controllers in CodeIgniter 3 Part 1

A controller is a class file which contains methods

http://localhost/<project-folder>/Controller-Class/Method-inside-Controller

Each and every controller class will be saved in: Application/Controllers

Class name and File name should be matched:

Class Employee { ... }

Employee.php

Class name should start wih Capital Letter:

Class Employee { ... }

Each and every controller class should extends with a base class called CI\_Controller

HTACCESS

code:

RewriteEngine On

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteCond %{REQUEST\_FILENAME} !-d

RewriteRule ^(.\*)$ index.php/$1 [L]

location to save the file: htdocs/<project-folder>/.htaccess

=====================================================================================================================================================

11- Controllers in CodeIgniter Part 2 | Remapping method Calls

public function \_remap($method/$action) {

if ($method/$action === 'some\_method/some\_action') {

$this -> $method()/$action();

}

else {

$this ->default\_method();

}

}

=====================================================================================================================================================

12- Views in CodeIgniter 3 || 13- Loading views in controller

Views

Part where uses HTML, CSS, Bootstrap

Each and every view class will be saved in: Application/Views

How can we load a view into Controller?

code:

$this->load->view("<view-file-name>");

=====================================================================================================================================================

14- Models in CodeIgniter | What is a Model | How to load a Model into a Controller

Models

A model always deals with db data

A model is a class file

Each and every model class will be saved in: Application/Models

Model class name and file name must be matched. Class Employee { ... } Employee.php

Class name should start wih Capital Letter: Class Employee { ... }

Each and every model class should extends with a base class called CI\_Model

How can we load a model into Controller?

code:

$this->load->model("<model-file-name>");

=====================================================================================================================================================

15- View Controller Model Communication

Interactions in MVC:

From Model: Model (Data) => Controller => View

From View: View (Data) => Controller => Model

Model and View don't communicate directly with each other, They use Controller as Intermediary for communication

=====================================================================================================================================================

16- Helpers in CodeIgniter | Loading Helpers into Controllers || 17- Working with Helpers

Helpers in CodeIgniter

What is a helper?

A helper is a collection of methods

Helper class usually are saved in: Application/helpers or in system/helpers

How can we load a helper into Controller?

code:

$this->load->helper("<helper-file-name>");

How to define a base URL: <project-folder>/Application/config/config.php

In the command -----> $config['base\_url'] = '';

You can write the address to be defined as base URL

=====================================================================================================================================================

18- Libraries in CodeIgniter

A library is a class file

A helper is a function

code:

Welcome library class

test()

hi()

sample()

$this->load->helper("<helper-file-name>");

$this->load->view("<view-file-name>");

$this->load->model("<model-file-name>");

BlogModel

- getAllBlogs()

$this->load->model("BlogModel");

$this->BlogModel->getAllBlogs();

form\_validation

$this->load->library("name");

$this->load->library("email");

$this->load->library("upload");

$this->load->library("pagination");

$this->load->library(array("email","upload","pagination"));

form\_validation

- run()

- set\_rules()

----------------------------------------

Loading libraries into controller

----------------------------------------

$this->load->library("form\_validation");

$this->form\_validation->run();

$this->form\_validation->set\_rules();

=====================================================================================================================================================

19- Email Library in CodeIgniter

$this->load->library("email");

$this->load->to("destination\_email");

$this->load->subject("subject");

$this->load->message("message");

$this->load->cc("cc");

$this->load->bcc("bcc");

=====================================================================================================================================================

20- CodeIgniter 3 Form Validation

Form validation

- set\_rules()

- run()

$this->load->library("form\_validation");

$this->form\_validation->set\_rules("name\_input\_field", "label\_display\_error\_msg", "rules");

rulse ex.: required|min\_length[3]|max\_length[25]

ex.:

$this->form\_validation->set\_rules("name", "Name", "required|min\_length[3]");

$this->form\_validation->set\_rules("email", "Email", "required|valid\_email");

$this->form\_validation->set\_rules("mobile", "Moble", "required|numeric");

echo form\_error("name\_input\_field");

=====================================================================================================================================================

21- Database Configuration in CodeIgniter 3

Database Configuration

- database.php

- Config/Database.php

Load database in Controller or Model:

$this->load->database();

- Controller DB

- Model1

- Model2

- Model3

mysqli\_error($con);

$this->db->query();

$this->db->error();

mysqli\_fetch\_row();

mysqli\_fetch\_assoc();

mysqli\_fetch\_array();

mysqli\_fetch\_object():

Fetching data from result set in CI

------------------------------------

result();

result\_object();

result\_array();

row();

row\_object();

row\_array();

=====================================================================================================================================================

22- Fetching data from table using MVC way

(Aula de Consulta)

=====================================================================================================================================================

23- QueryBuilder Class

- In CodeIgniter 3, the Query Builder is a convenient class used to interact with databases. It allows you to build and execute SQL queries in

a more structured, programmatic way, abstracting away raw SQL syntax. Below is an overview of how to use CodeIgniter 3's Query Builder.

- Example Setup

- First, ensure that the database is loaded in your config/autoload.php or through a controller.

// autoload.php

$autoload['libraries'] = array('database');

- Alternatively, load the database in your controller method.

// Inside a controller

$this->load->database();

- Basic Query Builder Usage

1. Select Query

- To retrieve data from a table, use the select() method to specify the columns and get() to execute the query.

// Selecting all records from a table

$query = $this->db->get('table\_name');

// Select specific columns

$this->db->select('id, name, email');

$query = $this->db->get('users');

2. Where Clause

- You can use the where() method to add conditions to your query.

// Single condition

$this->db->where('status', 'active');

$query = $this->db->get('users');

// Multiple conditions

$this->db->where('status', 'active');

$this->db->where('age >', 18);

$query = $this->db->get('users');

3. Insert Query

- To insert data, use the insert() method.

$data = array(

'name' => 'John Doe',

'email' => 'johndoe@example.com',

'status' => 'active'

);

$this->db->insert('users', $data);

4. Update Query

- To update data, use the update() method along with where() to specify which record to update.

$data = array(

'status' => 'inactive'

);

$this->db->where('id', 1);

$this->db->update('users', $data);

5. Delete Query

- For deleting records, you can use the delete() method.

$this->db->where('id', 1);

$this->db->delete('users');

6. Limit and Offset

- Use limit() and offset() to limit the number of rows returned or to implement pagination.

// Get the first 10 records

$query = $this->db->limit(10)->get('users');

// Get records 10 to 20

$query = $this->db->limit(10, 10)->get('users');

7. Order By

- Use order\_by() to specify the ordering of the result.

// Order by name in ascending order

$this->db->order\_by('name', 'asc');

$query = $this->db->get('users');

8. Join Query

- You can join multiple tables using join().

$this->db->select('users.id, users.name, orders.order\_date');

$this->db->from('users');

$this->db->join('orders', 'orders.user\_id = users.id');

$query = $this->db->get();

9. Like Clause

- To perform a LIKE search, use the like() method.

$this->db->like('name', 'John');

$query = $this->db->get('users');

10. Distinct Query

- To retrieve distinct values, you can use distinct().

$this->db->distinct();

$this->db->select('name');

$query = $this->db->get('users');

11. group\_by()

- Used to group rows by one or more columns (similar to SQL GROUP BY).

$this->db->select('category, COUNT(\*) as total');

$this->db->from('products');

$this->db->group\_by('category');

$query = $this->db->get();

12. having()

- Used to filter records after applying GROUP BY (similar to SQL HAVING).

$this->db->select('category, COUNT(\*) as total');

$this->db->from('products');

$this->db->group\_by('category');

$this->db->having('COUNT(\*) >', 5); // Products with more than 5 in category

$query = $this->db->get();

13. or\_where()

- Used to add an "OR" condition to your query. It works the same as where(), but applies the OR operator.

$this->db->where('status', 'active');

$this->db->or\_where('age >', 30);

$query = $this->db->get('users');

14. like()

- Performs a LIKE query, used for partial matching in text columns.

$this->db->like('name', 'John');

$query = $this->db->get('users');

- You can also specify whether the LIKE should be at the beginning (before), end (after), or both (both).

$this->db->like('name', 'John', 'after');

$query = $this->db->get('users');

15. not\_like()

- Performs a NOT LIKE query to exclude records that match the specified pattern.

$this->db->not\_like('name', 'John');

$query = $this->db->get('users');

16. or\_like()

- Similar to like(), but applies the "OR" condition.

$this->db->like('name', 'John');

$this->db->or\_like('email', 'example.com');

$query = $this->db->get('users');

17. or\_not\_like()

- Similar to not\_like(), but applies the "OR" condition.

$this->db->not\_like('name', 'John');

$this->db->or\_not\_like('email', 'example.com');

$query = $this->db->get('users');

18. in()

- Used to check if a column value is in a set of values (SQL IN clause).

$this->db->where\_in('status', array('active', 'pending'));

$query = $this->db->get('users');

19. not\_in()

- Used to exclude certain values from a query (SQL NOT IN clause).

$this->db->where\_not\_in('status', array('inactive', 'banned'));

$query = $this->db->get('users');

20. where\_in() and where\_not\_in() with subqueries

- You can also use subqueries within where\_in() and where\_not\_in().

$subquery = $this->db->select('id')->from('users')->where('status', 'active');

$this->db->where\_in('user\_id', $subquery);

$query = $this->db->get('orders');

21. exists()

- Checks whether a subquery exists. It's often used in conjunction with where() or other conditional methods.

$this->db->where('status', 'active');

$this->db->where('EXISTS (SELECT \* FROM orders WHERE orders.user\_id = users.id)');

$query = $this->db->get('users');

22. select\_max() and select\_min()

- Used to get the maximum or minimum value of a column.

$this->db->select\_max('price');

$query = $this->db->get('products');

// or for min

$this->db->select\_min('price');

$query = $this->db->get('products');

23. select\_avg() and select\_sum()

- To get the average (AVG) or sum (SUM) of a column.

$this->db->select\_avg('price');

$query = $this->db->get('products');

// or for sum

$this->db->select\_sum('quantity');

$query = $this->db->get('orders');

24. insert\_batch()

- Allows you to insert multiple rows in one query (batch insert).

$data = array(

array('name' => 'John', 'email' => 'john@example.com'),

array('name' => 'Jane', 'email' => 'jane@example.com')

);

$this->db->insert\_batch('users', $data);

25. update\_batch()

- Allows you to update multiple records at once (batch update).

$data = array(

array('id' => 1, 'status' => 'active'),

array('id' => 2, 'status' => 'inactive')

);

$this->db->update\_batch('users', $data, 'id');

26. delete\_batch()

- Used to delete multiple rows in one query (batch delete).

$ids = array(1, 2, 3);

$this->db->where\_in('id', $ids);

$this->db->delete('users');

27. trans\_start() and trans\_complete()

- These are used for transactions. You can group multiple database operations together and either commit or rollback them as a

single unit.

$this->db->trans\_start();

$this->db->insert('users', $data);

$this->db->insert('orders', $orderData);

$this->db->trans\_complete();

if ($this->db->trans\_status() === FALSE) {

// Transaction failed

}

28. escape()

- Escapes a string for safe inclusion in SQL queries.

$safe\_string = $this->db->escape('O\'Reilly');

29. get\_compiled\_select()

- Returns the SQL query string generated by the Query Builder for a SELECT statement.

$this->db->select('name, email');

$this->db->from('users');

$sql = $this->db->get\_compiled\_select();

echo $sql; // Outputs the SQL query string

30. get\_compiled\_insert()

- Similar to get\_compiled\_select(), but for INSERT queries.

$data = array('name' => 'John', 'email' => 'john@example.com');

$sql = $this->db->set($data)->get\_compiled\_insert('users');

echo $sql; // Outputs the SQL query string

21. count\_all()

Counts all the rows in a table.

$count = $this->db->count\_all('users');

echo $count; // Outputs the number of rows in the 'users' table

22. count\_all\_results()

Returns the number of rows that match a particular query, like SELECT COUNT(\*).

$this->db->where('status', 'active');

$count = $this->db->count\_all\_results('users');

echo $count; // Outputs the number of active users

23. set()

Used to set individual values for an UPDATE statement. Can be used before update().

$this->db->set('status', 'inactive');

$this->db->where('id', 1);

$this->db->update('users');

24. select() with array()

You can pass an array of columns for the select() method.

$this->db->select(array('id', 'name', 'email'));

$query = $this->db->get('users');

25. truncate()

Truncates a table (removes all rows) but does not delete the table structure.

$this->db->truncate('users');

26. escape\_like\_str()

Escapes a string for use in a LIKE query. This is used to prevent issues with special characters when searching for substrings.

$search\_term = "O'Reilly";

$safe\_term = $this->db->escape\_like\_str($search\_term);

$this->db->like('name', $safe\_term);

$query = $this->db->get('users');

27. list\_fields()

Returns a list of all field names from a table.

$fields = $this->db->list\_fields('users');

print\_r($fields); // Array of column names in 'users' table

28. field\_data()

Returns metadata about the fields in a table.

$fields = $this->db->field\_data('users');

print\_r($fields); // Array of field metadata (type, length, etc.)

29. trans\_begin()

Starts a database transaction manually, useful if you want to manage transactions without automatic start and end.

$this->db->trans\_begin();

$this->db->insert('users', $data);

$this->db->insert('orders', $orderData);

if ($this->db->trans\_status() === FALSE) {

// Transaction failed, perform rollback

$this->db->trans\_rollback();

} else {

// Transaction succeeded, commit changes

$this->db->trans\_commit();

}

30. trans\_rollback()

Rolls back a transaction if there is an error during a transaction.

$this->db->trans\_rollback();

31. trans\_commit()

Commits the transaction, finalizing the changes made.

$this->db->trans\_commit();

32. db\_debug

A property that can be set to FALSE to suppress error output in case of failure (useful for production).

$this->db->db\_debug = FALSE;

33. set\_dbdriver()

Allows you to change the database driver dynamically.

$this->db->set\_dbdriver('mysqli');

34. insert\_id()

Returns the last inserted ID after an INSERT query. This is useful for auto-increment fields.

$this->db->insert('users', $data);

$last\_id = $this->db->insert\_id();

echo $last\_id; // Last inserted ID

35. affected\_rows()

Returns the number of affected rows from the last query, typically used for INSERT, UPDATE, or DELETE operations.

$this->db->update('users', $data, array('id' => 1));

echo $this->db->affected\_rows(); // Number of rows updated

36. trans\_status()

Checks the status of the current transaction (whether it was successful or not).

$status = $this->db->trans\_status();

if ($status === FALSE) {

// Transaction failed

}

37. cache\_on() and cache\_off()

These methods allow you to enable or disable query caching for the current database session. Query results are cached for faster subsequent retrievals.

$this->db->cache\_on(); // Enable caching

$query = $this->db->get('users');

// Disable caching

$this->db->cache\_off();

38. cache\_delete()

Deletes the cached query result for a specific table.

$this->db->cache\_delete('users');

39. query() with binds

If you want to execute a query with bound parameters (like in prepared statements), use query() along with binds.

$sql = "SELECT \* FROM users WHERE id = ?";

$query = $this->db->query($sql, array($user\_id));

40. insert\_string()

Returns the INSERT query string, without actually executing the query. Useful if you need to examine the generated SQL.

$data = array('name' => 'John', 'email' => 'john@example.com');

$sql = $this->db->insert\_string('users', $data);

echo $sql; // Outputs the SQL string

41. update\_string()

Returns the UPDATE query string, without actually executing the query.

$data = array('status' => 'inactive');

$sql = $this->db->update\_string('users', $data, array('id' => 1));

echo $sql; // Outputs the SQL string

42. delete\_string()

Returns the DELETE query string, without actually executing the query.

$sql = $this->db->delete\_string('users', array('id' => 1));

echo $sql; // Outputs the SQL string

43. protect\_identifiers()

Escapes table names, column names, or database identifiers to protect against SQL injection and reserved keywords.

$table = $this->db->protect\_identifiers('users');

44. escape()

Escapes a string to be safely used in queries (particularly useful for dealing with special characters in strings).

$safe\_string = $this->db->escape("O'Reilly");

- Example: Complex Query Using Multiple Methods

- Here’s an example of using various methods in combination to fetch data with a join, where condition, limit, and ordering.

$this->db->select('users.id, users.name, users.email, orders.order\_date');

$this->db->from('users');

$this->db->join('orders', 'orders.user\_id = users.id');

$this->db->where('users.status', 'active');

$this->db->order\_by('orders.order\_date', 'desc');

$this->db->limit(10);

$query = $this->db->get();

- Raw Queries

- Sometimes, you may need to run raw queries that aren’t covered by the Query Builder methods. In this case, you can use the query()

method.

$sql = "SELECT \* FROM users WHERE status = 'active'";

$query = $this->db->query($sql);

- Checking for Errors

- After executing queries, you can check for errors using error().

if ($this->db->trans\_status() === FALSE) {

// Handle error

$error = $this->db->error();

echo $error['message'];

}

- Conclusion

- CodeIgniter 3’s Query Builder provides a powerful and flexible way to interact with your database without writing raw SQL queries.

It helps with writing cleaner, safer, and more maintainable code.

=====================================================================================================================================================

24- Registration form part1

(Aula de Consulta / Aula prática)

=====================================================================================================================================================

25- Registration form part2 | Saving data into a table

(Aula de Consulta / Aula prática)

=====================================================================================================================================================

26- Registration form with account activation process | Send an email to registered user

(Aula de Consulta / Aula prática)

=====================================================================================================================================================

27- Theme Integration in CodeIgniter

(Aula de Consulta / Aula Prática)

=====================================================================================================================================================

28- Login Form Part1

(Aula de Consulta / Aula Prática)

=====================================================================================================================================================

29- Login Form Part2

(Aula de Consulta / Aula Prática)

=====================================================================================================================================================

30- Logout functionality

(Aula de Consulta / Aula Prática)

=====================================================================================================================================================

31- Welcome profile of logged in user

(Aula de Consulta / Aula Prática)

=====================================================================================================================================================

32- Change password

(Aula de Consulta / Aula Prática)

=====================================================================================================================================================

33- Edit profile logged in user

(Aula de Consulta / Aula Prática)