



Assignment Module 17

Full Name: **Sakir Ahamed Bissas**

Phone: **01995542277**

Date of Submission: **12 Jun 2023**

<https://github.com/sakirgit/Ostad-Assignment-module-17-laravel>

Question: 1

Explain what Laravel's query builder is and how it provides a simple and elegant way to interact with databases.

Laravel's query builder is a feature that simplifies interacting with databases by providing a fluent interface for constructing and executing queries. It abstracts SQL syntax and offers methods for common database operations. Here's an example of using the query builder to retrieve all users from the "users" table:

```
$users = DB::table('users')
    ->select('name', 'email')
    ->where('age', '>', 25)
    ->orderBy('name')
    ->get();
```

In this example, we use the `DB` facade to access the query builder. We call the `table` method to specify the "users" table.

Next, we chain the `select` method to specify the columns to retrieve (`name` and `email` in this case).

We add a `where` condition to filter the users based on their age, selecting only those with an age greater than 25.

Then, we use the `orderBy` method to sort the users by their name.

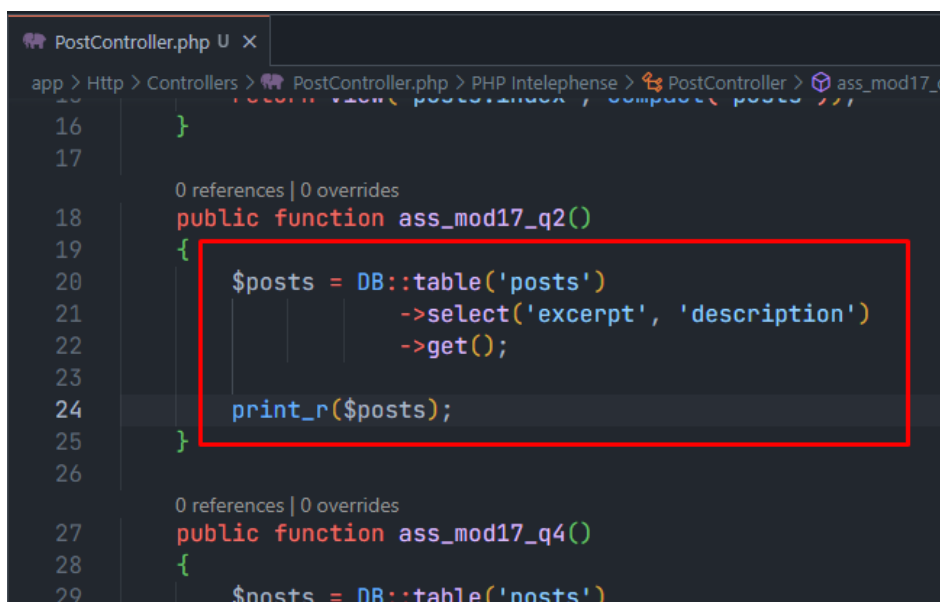
Finally, we call the `get` method to execute the query and retrieve the resulting records as a collection, which is stored in the `\$users` variable.

This is just a basic example, but the query builder offers many more methods and functionalities to handle more complex queries and operations.

Overall, the query builder provides a simple and elegant way to interact with databases in Laravel, allowing you to build queries fluently and readably, without the need to write raw SQL statements.

Question: 2

Write the code to retrieve the "excerpt" and "description" columns from the "posts" table using Laravel's query builder. Store the result in the \$posts variable. Print the \$posts variable.



```
PostController.php U X
app > Http > Controllers > PostController.php > PHP Intelephense > PostController > ass_mod17_q2
16     }
17
18     0 references | 0 overrides
19     public function ass_mod17_q2()
20     {
21         $posts = DB::table('posts')
22             ->select('excerpt', 'description')
23             ->get();
24         print_r($posts);
25     }
26
27     0 references | 0 overrides
28     public function ass_mod17_q4()
29     {
30         $posts = DB::table('posts')
```

Question: 3

Describe the purpose of the `distinct()` method in Laravel's query builder. How is it used in conjunction with the `select()` method?

The `distinct()` method in Laravel's query builder is used to retrieve only unique values from a specific column or set of columns in the database table. It ensures that duplicate values are eliminated, and only distinct values are returned in the result set.

When used in conjunction with the `select()` method, `distinct()` is placed before `select()` to indicate that the query should retrieve distinct values for the specified columns. Here's a short example:

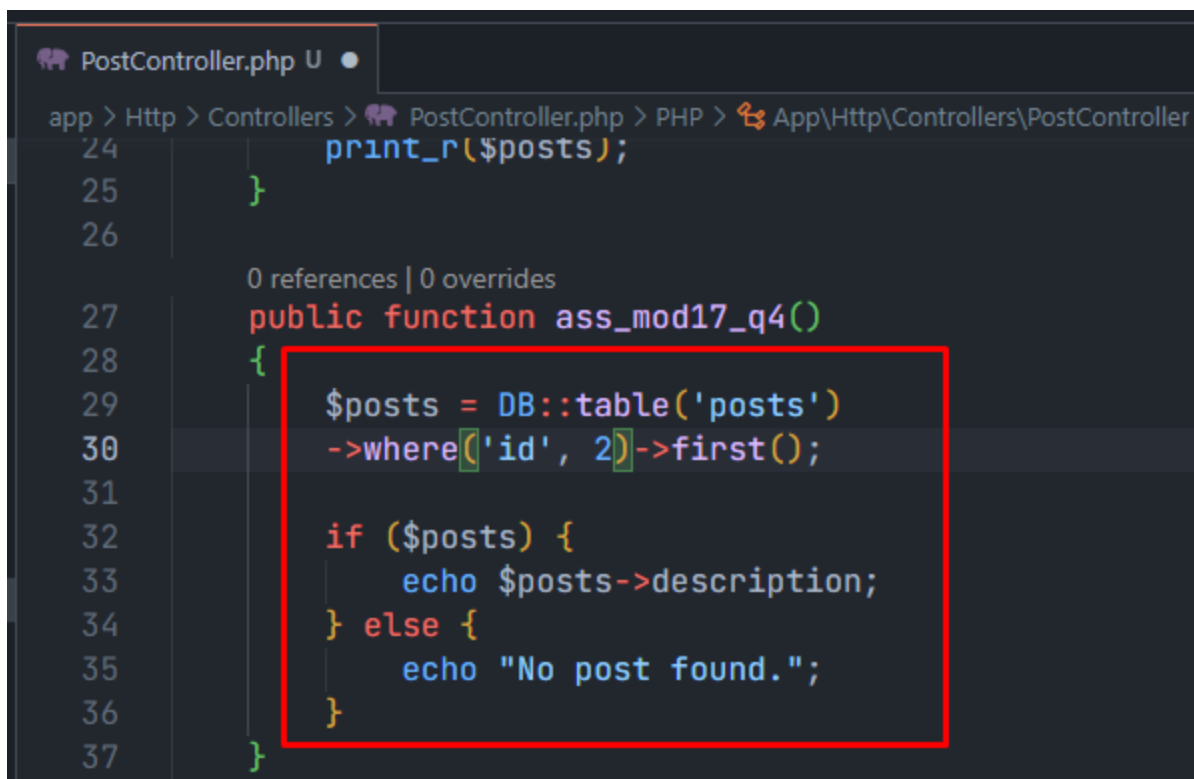
```
$emails = DB::table('users')
    ->select('email')
    ->distinct()
    ->get();
```

In this example, the query builder selects the "email" column from the "users" table. The `distinct()` method is called to ensure that only distinct email addresses are returned in the result set. This means that if there are any duplicate email addresses in the "users" table, only one of each unique email address will be included in the final result.

Using `distinct()` in conjunction with `select()` allows you to retrieve unique values from specific columns while ignoring any duplicates. It is useful when you want to eliminate repetitive values and focus on unique entries in your query results.

Question: 4

Write the code to retrieve the first record from the "posts" table where the "id" is 2 using Laravel's query builder. Store the result in the `$posts` variable. Print the "description" column of the `$posts` variable.



```
PostController.php U ●
app > Http > Controllers > PostController.php > PHP > App\Http\Controllers\PostController
24     print_r($posts);
25 }
26
    0 references | 0 overrides
27 public function ass_mod17_q4()
28 {
29     $posts = DB::table('posts')
30     ->where('id', 2)->first();
31
32     if ($posts) {
33         echo $posts->description;
34     } else {
35         echo "No post found.";
36     }
37 }
```

Question: 5

Write the code to retrieve the "description" column from the "posts" table where the "id" is 2 using Laravel's query builder. Store the result in the \$posts variable. Print the \$posts variable.



```
PostController.php U
app > Http > Controllers > PostController.php > PHP > App\Http\Controllers\Post
37     }
38
    0 references | 0 overrides
39     public function ass_mod17_q5()
40     {
41         $posts = DB::table('posts')
42             ->where('id', 2)
43             ->select('description')
44             ->first();
45
46         dd($posts);
47     }
    0 references | 0 overrides
49     public function ass_mod17_q7()
```

Question: 6

Explain the difference between the first() and find() methods in Laravel's query builder. How are they used to retrieve single records?

Difference between first() and find() in Laravel's query builder:

first(): retrieves the first record that matches the query conditions, returning an instance of stdClass or null if no match is found. It is used when you want to retrieve a single record based on specific conditions.

find(): retrieves a record by its primary key value. It assumes the primary key column is named "id" and returns a single model instance or null if no record is found. It is used when you want to retrieve a record by its primary key.

Question: 7

Write the code to retrieve the "title" column from the "posts" table using Laravel's query builder. Store the result in the \$posts variable. Print the \$posts variable.

```
PostController.php U ●
app > Http > Controllers > PostController.php > PHP > App\Http\Controllers\PostController.php
48
0 references | 0 overrides
49 public function ass_mod17_q7()
50 {
51     $posts = DB::table('posts')
52     ->select('title')
53     ->get();
54
55     foreach ($posts as $post) {
56         echo $post->title . PHP_EOL;
57     }
58 }
59
0 references | 0 overrides
60 public function ass_mod17_q8()
```

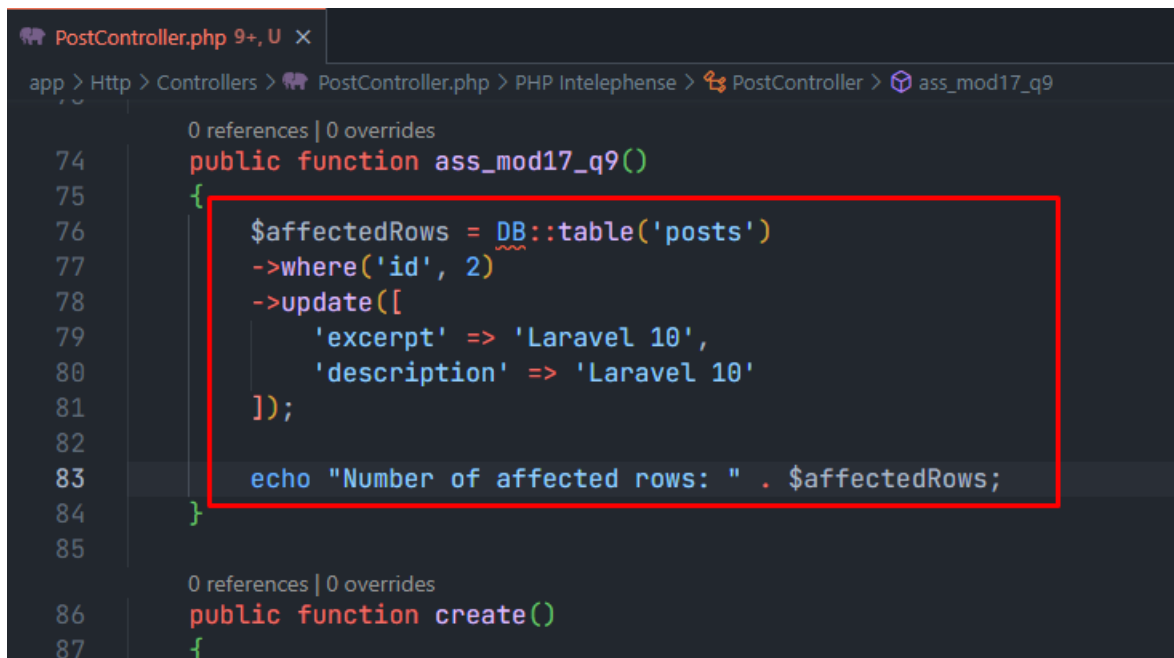
Question: 8

Write the code to insert a new record into the "posts" table using Laravel's query builder. Set the "title" and "slug" columns to 'X', and the "excerpt" and "description" columns to 'excerpt' and 'description', respectively. Set the "is_published" column to true and the "min_to_read" column to 2. Print the result of the insert operation.

```
PostController.php U ●
app > Http > Controllers > PostController.php > PHP > App\Http\Controllers\PostController.php
0 references | 0 overrides
60 public function ass_mod17_q8()
61 {
62     $result = DB::table('posts')->insert([
63         'title' => 'X',
64         'slug' => 'X',
65         'excerpt' => 'excerpt',
66         'description' => 'description',
67         'is_published' => true,
68         'min_to_read' => 2
69     ]);
70
71     print_r($result);
72 }
73 }
```

Question: 9

Write the code to update the "excerpt" and "description" columns of the record with the "id" of 2 in the "posts" table using Laravel's query builder. Set the new values to 'Laravel 10'. Print the number of affected rows.



The screenshot shows a code editor with the file `PostController.php` open. The breadcrumb navigation indicates the path: `app > Http > Controllers > PostController.php > PHP Intelephense > PostController > ass_mod17_q9`. The code defines a public function `ass_mod17_q9()` starting at line 74. The function body is enclosed in a red box and contains the following logic: it initializes `$affectedRows` by calling `DB::table('posts')`, then filters the records with `->where('id', 2)`, performs the update with `->update(['excerpt' => 'Laravel 10', 'description' => 'Laravel 10'])`, and finally prints the number of affected rows with `echo "Number of affected rows: " . $affectedRows;`. The function ends at line 84. Below it, the start of another function `create()` is visible at line 86.

```
74 public function ass_mod17_q9()
75 {
76     $affectedRows = DB::table('posts')
77     ->where('id', 2)
78     ->update([
79         'excerpt' => 'Laravel 10',
80         'description' => 'Laravel 10'
81     ]);
82
83     echo "Number of affected rows: " . $affectedRows;
84 }
85
86 public function create()
87 {
```

Question 10:

Write the code to delete the record with the "id" of 3 from the "posts" table using Laravel's query builder. Print the number of affected rows.



The screenshot shows the same code editor with the file `PostController.php` open. The breadcrumb navigation indicates the path: `app > Http > Controllers > PostController.php > PHP Intelephense > PostController > ass_mod17_q10`. The code defines a public function `ass_mod17_q10()` starting at line 86. The function body is enclosed in a red box and contains the following logic: it initializes `$affectedRows` by calling `DB::table('posts')`, filters the records with `->where('id', 3)`, performs the deletion with `->delete();`, and finally prints the number of affected rows with `echo "Number of affected rows: " . $affectedRows;`. The function ends at line 93. Below it, the start of another function `create()` is visible at line 95.

```
84 }
85
86 public function ass_mod17_q10()
87 {
88     $affectedRows = DB::table('posts')
89     ->where('id', 3)
90     ->delete();
91
92     echo "Number of affected rows: " . $affectedRows;
93 }
94
95 public function create()
96 {
97     return view('posts.create');
```

Question 11:

Explain the purpose and usage of the aggregate methods `count()`, `sum()`, `avg()`, `max()`, and `min()` in Laravel's query builder. Provide an example of each.

1. `count()`: The `count()` method is used to retrieve the number of records in a table or the number of records that match specific conditions.

Example:

```
$totalPosts = DB::table('posts')->count();
```

This code retrieves the total number of records in the "posts" table.

2. `sum()`: The `sum()` method calculates the sum of the values in a specific column.

Example:

```
$totalAmount = DB::table('orders')->sum('amount');
```

This code calculates the sum of the "amount" column in the "orders" table.

3. `avg()`: The `avg()` method calculates the average value of a specific column.

Example:

```
$averagePrice = DB::table('products')->avg('price');
```

This code calculates the average price of the products in the "products" table.

4. `max()`: The `max()` method retrieves the maximum value of a specific column.

Example:

```
$highestScore = DB::table('scores')->max('score');
```

This code retrieves the highest score from the "scores" table.

5. `min()`: The `min()` method retrieves the minimum value of a specific column.

Example:

```
$lowestTemperature = DB::table('weather')->min('temperature');
```

This code retrieves the lowest recorded temperature from the "weather" table.

Question 12

Describe how the `whereNot()` method is used in Laravel's query builder. Provide an example of its usage.

- The `whereNot()` method is used to exclude records that satisfy a specific condition from the result set.

- It works in conjunction with the `where()` method to add a "not" condition to a query.

For example, let's say we have a "users" table with a "status" column, and we want to retrieve all the users who are not in an "active" status. We can use the `whereNot()` method as follows:

```
$users = DB::table('users')
    ->whereNot('status', 'active')
    ->get();
```

In the code above, we use the `DB` facade to access the query builder. We call the `table` method to specify the "users" table.

The `whereNot()` method is used to add a condition where the "status" column should not be equal to 'active'. This condition excludes users who have an "active" status from the result.

Finally, the `get()` method is called to execute the query and retrieve the resulting users as a collection.

With this code, we can retrieve all the users who are not in an "active" status, such as users with "inactive" or "suspended" statuses.

The `whereNot()` method provides a concise way to exclude records that meet a certain condition, allowing us to filter out unwanted results based on specific criteria.

Question 13

Explain the difference between the `exists()` and `doesntExist()` methods in Laravel's query builder. How are they used to check the existence of records?

- `exists()`: It checks if any records exist in a table that satisfy a specific condition. Returns `true` if any records match the condition, and `false` otherwise.

Example:

```
$exists = DB::table('users')
    ->where('status', 'active')
    ->exists();
```

In this example, `$exists` will be `true` if there is at least one "active" user in the "users" table. Otherwise, it will be `false`.

- `doesntExist()`: It checks if no records exist in a table that satisfy a specific condition. Returns `true` if no records match the condition, and `false` if any records are found.

Example:

```
$doesntExist = DB::table('users')
    ->where('status', 'inactive')
    ->doesntExist();
```

In this example, `$doesntExist` will be `true` if there are no "inactive" users in the "users" table. Otherwise, it will be `false`.

In summary, `exists()` checks if any records match the condition, while `doesntExist()` checks if no records match the condition. These methods provide a simple way to determine whether records exist or not based on specific criteria.

Please note that the actual implementation of `whereNot()` may involve more complex conditions or operators depending on your specific needs.

Question 14

Write the code to retrieve records from the "posts" table where the "min_to_read" column is between 1 and 5 using Laravel's query builder. Store the result in the \$posts variable. Print the \$posts variable.

```
PostController.php U X
app > Http > Controllers > PostController.php > PHP Intelephense > PostController > ass_m
96
0 references | 0 overrides
97 public function ass_mod17_q14()
98 {
99     $posts = DB::table('posts')
100     ->whereBetween('min_to_read', [1, 5])
101     ->get();
102
103     print_r($posts);
104 }
105
0 references | 0 overrides
106 public function create()
107 {
108     return view('posts.create');
```

Question 15

Write the code to increment the "min_to_read" column value of the record with the "id" of 3 in the "posts" table by 1 using Laravel's query builder. Print the number of affected rows.

```
PostController.php U X
app > Http > Controllers > PostController.php > PHP Intelephense > PostController > ass_mod17_q15
104 }
105
106
0 references | 0 overrides
107 public function ass_mod17_q15()
108 {
109     $affectedRows = DB::table('posts')
110     ->where('id', 3)
111     ->increment('min_to_read', 1);
112
113     echo "Number of affected rows: " . $affectedRows;
114 }
115
0 references | 0 overrides
116 public function create()
117 {
```