Module 17 Assignment

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

Answer:

Laravel's query builder is a powerful feature of the Laravel framework that allows developers to build and execute database queries using a fluent and intuitive API. It provides a simple and elegant way to interact with databases by abstracting the complexities of writing raw SQL queries.

Fluent Interface: The query builder uses a fluent interface, which means that each method call returns an instance of the query builder itself. This allows developers to chain multiple methods together, creating readable and concise queries. For example:

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

Expressive Syntax: Laravel's query builder provides a rich set of methods that closely resemble SQL syntax, making it easy to construct queries. Developers can use methods like select, where, orderBy, groupBy, join, and more to specify the desired operations. This syntax is designed to be intuitive and familiar to developers who are already familiar with SQL.

Advanced Features: In addition to the basic query building capabilities, Laravel's query builder offers advanced features like aggregate functions, subqueries, unions, and raw expressions. These features provide developers with the flexibility to handle complex scenarios and perform advanced database operations using the query builder.

Overall, Laravel's query builder provides a clean and expressive way to interact with databases, abstracting the intricacies of SQL while still offering the flexibility and power to handle various database operations. It simplifies the process of building queries, improves code readability, and enhances the overall development experience in Laravel applications.

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.

Answer:

```
$posts = DB::table('posts')
            ->select('excerpt', 'description')
            ->get();

print_r($posts);
```

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

Answer:

The distinct() method in Laravel's query builder is used to retrieve only unique rows from a table based on the specified column(s). It ensures that duplicate rows are eliminated from the query result.

When used in conjunction with the select() method, the distinct() method allows you to select unique values from the specified column(s) in the query result.

Here's an example to illustrate its usage:

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

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.

Answer:

```
$posts = DB::table('posts')
           ->where('id', 2)
           ->first();

echo $posts->description;
```

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.

Answer:

```
$posts = DB::table('posts')
           ->where('id', 2)
           ->pluck('description');

print_r($posts)
```

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

Answer:

In Laravel's query builder, both the first() and find() methods are used to retrieve a single record from the database. However, they differ in their approach and usage:

first(): The first() method is used to retrieve the first record that matches the query criteria. It returns a single object representing the first matching row.

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

find(): The find() method is used to retrieve a single record based on its primary key value. It accepts the primary key value as an argument and returns a single object representing the matching row.

```
$record = DB::table('users')->find(1);
```

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.

Answer:

```
$posts = DB::table('posts')
            ->select('title')
            ->get();


print_r($posts);
```

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.

Answer:

```
$result = DB::table('posts')->insert([
    'title' => 'X',
    'slug' => 'X',
    'excerpt' => 'excerpt',
    'description' => 'description',
    'is_published' => true,
    'min_to_read' => 2,
]);

print_r($result);
```

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.

Answer:

```
$affectedRows = DB::table('posts')
        ->where('id', 2)
        ->update([
            'excerpt' => 'Laravel 10',
            'description' => 'Laravel 10'
        ]);

echo "Number of affected rows: " . $affectedRows;
```

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.

Answer:

```
$affectedRows = DB::table('posts')
        ->where('id', 3)
        ->delete();

echo "Number of affected rows: " . $affectedRows;
```

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.

Answer:

count(): The count() method is used to retrieve the number of rows matching the query criteria.

$count = DB::table('users')->count();

In this example, the count() method retrieves the total number of rows in the "users" table

sum(): The sum() method is used to calculate the sum of the values in a specific column.

$total = DB::table('sales')->sum('amount');

In this example, the sum('amount') method calculates the total sum of the "amount" column in the "sales" table.

avg(): The avg() method is used to calculate the average value of a specific column.

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

In this example, the avg('price') method calculates the average price of the products stored in the "products" table.

max(): The max() method is used to retrieve the maximum value from a specific column.

$maximum = DB::table('scores')->max('points');

In this example, the max('points') method retrieves the highest score (maximum value) from the "points" column in the "scores" table.

min(): The min() method is used to retrieve the minimum value from a specific column.

```
$minimum = DB::table('temperatures')->min('degrees');
```

In this example, the min('degrees') method retrieves the lowest temperature (minimum value) from the "degrees" column in the "temperatures" table.

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

Answer:

In Laravel's query builder, the whereNot() method is used to add a "not" condition to the query. It allows you to exclude rows that match a specific condition from the result set.

The whereNot() method takes two arguments: the column name and the value to compare against. It adds a WHERE condition to the query, stating that the column value should not be equal to the provided value.

Here's an example to illustrate the usage of whereNot():

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

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?

Answer:

exists(): The exists() method is used to check if any records exist in the result set of a query**.**

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

doesntExist(): The doesntExist() method is used to check if no records exist in the result set of a query.

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

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.

Answer:

```
$posts = DB::table('posts')
        ->whereBetween('min_to_read', [1, 5])
        ->get();

print_r($posts);
```

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.

Answer:

```
$affectedRows = DB::table('posts')
                ->where('id', 3)
                ->increment('min_to_read', 1);

echo "Number of affected rows: " . $affectedRows;
```