Assignment - 17: Laravel Query Builder

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

Ans:

Laravel's query builder is a feature of the Laravel framework that provides a fluent and intuitive interface for interacting with databases. It allows you to build and execute database queries using a chainable method syntax, making database operations more concise and readable. The query builder abstracts the underlying database operations, providing a consistent API across different database systems.

Here are some key points that highlight how Laravel's query builder offers a simple and elegant way to interact with databases:

Fluent Method Chaining:
The query builder utilizes method chaining, allowing user to express complex database queries in a more readable and expressive manner. User can chain multiple methods together to build queries step-by-step, enhancing code clarity.

Parameter Binding:
Laravel's query builder handles parameter binding automatically, protecting against SQL injection attacks. User can pass parameters to the query builder methods without worrying about escaping or sanitizing them manually.

Database Agnostic:
The query builder is designed to work with various database systems supported by Laravel, including MySQL, PostgreSQL, SQLite, and SQL Server. It abstracts the differences in syntax and functionality and allow to write database-agnostic queries.

Eloquent ORM Integration:
The query builder seamlessly integrates with Laravel's Eloquent ORM, providing a unified interface for both simple queries and complex model relationships.User can switch between the query builder and Eloquent ORM methods effortlessly.

Query Optimization:

Laravel's query builder includes optimization features such as query caching, eager loading, and lazy loading. These features help improve performance by reducing the number of database queries and optimizing data retrieval.

Query Logging and Debugging:
Laravel's query builder allows log and debug queries easily. User can enable query logging to view the executed queries, their bindings, and the time taken for execution, which aids in troubleshooting and optimizing database interactions.

Extensibility:
Laravel's query builder is highly extensible. This allows to create custom query builder macros and extend its functionality according to your specific needs.User can encapsulate common query patterns into reusable methods, promoting code reuse and maintainability.

Overall, Laravel's query builder provides a clean and intuitive API for interacting with databases, offering a simplified and elegant approach to performing database operations. It simplifies the process of writing and executing database queries, resulting in more readable and maintainable code.

(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.

Ans:

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

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

Ans:

The distinct() method in Laravel's query builder is used to retrieve unique values from a column in a database table. It ensures that the result set contains only distinct (unique) values for the specified column.

When used in conjunction with the select() method, the distinct() method modifies the query to return distinct values for the specified column(s) in the SELECT statement.

Here's an example to illustrate its usage:

```
$uniqueCategories = DB::table('products')
  ->select('category')
  ->distinct()
  ->get();
```

In the above code, the select('category') method specifies that we want to retrieve the "category" column from the "products" table. By chaining the distinct() method, we ensure that only distinct values of the "category" column are returned in the result set.

(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.

Ans:

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

if ($posts) {
    echo $posts->description;
} else {
    echo 'No record found.';
}
```

(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.

Ans:

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

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

Ans:

Both the first() and find() methods are used to retrieve a single record from the database. However, there are differences in how they work and the conditions under which they are used.

difference between the first() and find():
The key difference between first() and find() is the way they retrieve records. The first() method is used to retrieve the first record that matches the query conditions, whereas the find() method retrieves a record by its id column value.

first() method's working process:
The first() method is used to retrieve the first record that matches the query conditions. It returns a single model instance (an object) representing the first record found. It is commonly used when we want to retrieve the first result from a query result set. For example,

$user = DB::table('users')->where('age', '>', 50)->first();

In the above example, the first() method is used to retrieve the first user who is above 50 years old. The method retrieves the first matching record based on the query conditions and returns a single instance of the model.

find() method's working process:
The find() method is used to retrieve a single record by its primary key. It expects the primary key value as an argument and returns the corresponding record. For example,

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

In the above example, the find(1) method retrieves the user which id column value equals to 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.

Ans:

The following code retrieve the "title" column from the "posts" table and store the result in $post variable. Finally, print_r($posts) is used to print the contents of the $posts variable, which will display an array of all the "title" values from the "posts" table.

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

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.

Ans:

```
$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.

Ans:

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

  print_r($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.

Ans:

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

print_r($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.

Ans:

The aggregate methods count(), sum(), avg(), max(), and min() are used to perform calculations on specific columns or expressions in a database table.

Here's an explanation of each aggregate method along with an example:

cout() :

The count() method is used to retrieve the total number of records matching the query conditions.

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

Here the count() method is used to find out th total number of rows in the users table.

sum() :

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

```
$totalAmount = DB::table('users')->sum('income');
```

In the above example, sum('income') calculates the total sum of the values in ‘income’ column in "users" table.

avg() :

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

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

In the above example, avg('price') calculates the average value of the "price" column in the "products" table.

max() :

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

```
$maxPrice = DB::table('products')->max('price');
```

In the above example, max('price') retrieves the maximum value from the "price" column in the "products" table.

min() :

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

```
$minPrice = DB::table('products')->min('price');
```

In the above example, min('price') retrieves the minimum value from the "price" column in the "products" table.

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

Ans:

In Laravel's query builder, the whereNot() method is used to add a "not" condition to a query. It allows to specify a column and a value that should not match in the result set. The whereNot() method is the opposite of the where() method.

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

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

In the above example, the whereNot() method is used to retrieve all the users whose status is not "active" from the "users" table.

(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?

Ans:

In Laravel's query builder, the exists() and doesntExist() methods are used to check the existence of records in a table. Here's an explanation of the difference between these methods and how they are used:

exists():
The exists() method is used to check if there are any records matching the query conditions. It returns true if at least one record exists; otherwise, it returns false.

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

In the above example, exists() is used to check if there is any user with a status of "active" in the "users" table. The method returns true if such a record exists; otherwise, it returns false.

doesntExist():

The doesntExist() method is the inverse of the exists() method. It checks if there are no records matching the query conditions. It returns true if no records exist; otherwise, it returns false.

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

In the above example, doesntExist() is used to check if there are no users with a status of "deleted" in the "users" table. The method returns true if no such records exist; otherwise, it returns false.

(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.

Ans:

```
$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.

Ans:

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

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