মডিউল ১৭ এর এসাইনমেন্ট

Assignment: Query Builder in Laravel

Questions:

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

Here are a few key aspects of Laravel's query builder that contribute to its simplicity and elegance:

Fluent Interface: The query builder utilizes a fluent interface, which means that each method call returns an instance of the query builder itself. This allows you to chain multiple methods together, creating a more readable and expressive syntax. For example, you can write queries like $users = DB::table('users')->where('active', true)->orderBy('name')->get();, where each method call adds a specific clause or condition to the query.

Method Chaining: Method chaining in the query builder allows you to build complex queries in a concise manner. You can chain methods like where(), orderBy(), join(), and more to construct queries that meet your requirements. This approach eliminates the need for manually concatenating SQL statements, resulting in cleaner and more maintainable code.

Parameter Binding: Laravel's query builder automatically handles parameter binding, which helps prevent SQL injection attacks. You can use placeholders in your queries, and the query builder will properly escape and sanitize the input values, ensuring the security and integrity of your data.

Database Agnostic: The query builder is designed to work with multiple database systems, including MySQL, PostgreSQL, SQLite, and SQL Server. It provides a consistent API for interacting with databases, regardless of the specific database engine you are using.

Query Optimization: Laravel's query builder includes various methods and functionalities to optimize database queries. You can use features like eager loading, caching, and query logging to improve the performance of your application and reduce the number of database trips.

ORM Integration: Laravel's query builder seamlessly integrates with Laravel's Object-Relational Mapping (ORM) tool, Eloquent. You can switch between using the query builder directly

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.

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

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

The distinct() method in Laravel's query builder is used to retrieve only unique values from a specified column or set of columns in the database query results.

When used in conjunction with the select() method, the distinct() method ensures that only unique values are returned for the selected columns.

Here's an example to illustrate the usage of distinct() with select() in Laravel's query builder:

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

the select() method, and distinct() will ensure the uniqueness of the combined values for those columns.

```
$tableNamesAndEmails = DB::table('users')->select('name',
'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.

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

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

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

first(): The first() method retrieves the first record that matches the specified conditions from the query. It returns an instance of the stdClass class, representing a single row of data from the table.

```
$post = DB::table('posts')->where('category', 'news')->first();
```

find(): The find() method retrieves a single record from the table based on its primary key. It specifically targets a record by its primary key value and returns an instance of the stdClass class.

```
$post = DB::table('posts')->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.

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

```
$result = DB::table('posts')->insert([
    'title' => 'X',
    'slug' => 'X',
```

```
    'excerpt' => 'excerpt',
    'description' => 'description',
    'is_published' => true,
    'min_to_read' => 2
]);

echo $result ? 'Record inserted successfully' : 'Failed to insert record';
```

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.

```
$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.

```
$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.

count(): The count() method is used to calculate the total number of rows that match a given condition. It returns the count as an integer value.

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

sum(): The sum() method calculates the sum of a column's values. It is commonly used with numeric columns such as "price" or "quantity" to determine the total value.

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

avg(): The avg() method calculates the average value of a numeric column. It returns the average as a floating-point value.

```
$averageRating = DB::table('reviews')->avg('rating');
```

max(): The max() method retrieves the maximum value from a column.

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

min(): The min() method retrieves the minimum value from a column.

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

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

whereNot() method is used to add a "not equal" condition to a query. It allows you to retrieve records where a specific column's value is not equal to a given value.

```
->whereNot('column', 'value');
```

Here's an example of how to use the whereNot() method in Laravel's query builder:

```
$users = DB::table('users')
    ->whereNot('status', 'inactive')
    ->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?

exists(): The exists() method is used to check if any records exist in the result set of a query. It returns a boolean value of true if at least one record is found, and false if the result set is empty.

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

doesntExist(): The doesntExist() method is the opposite of exists(). It is used to check if no records exist in the result set of a query. It returns a boolean value of true if the result set is empty, indicating that no records are found, and false if at least one record exists.

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

```
$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.

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

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