Assignment: Query Builder in Laravel

Name- Shah Obayed Ahmed

Contact- +8801749501291

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

Answer-

The query builder in Laravel is a robust tool that gives developers the ability to connect with databases through the use of a fluent and expressive `API`. It frees users from the necessity of writing raw `SQL` statements in order to construct and run database queries, making the process much simpler and more aesthetically pleasing. The query builder is one of the many tools that are included in the Laravel framework, which is a well-known `PHP` framework for the construction of web applications.

Developers are able to carry out a wide variety of database operations with the help of the query builder. These operations include fetching records, inserting data, updating records, and deleting data. Because it is compatible with a variety of database management systems, including `MySQL`, `PostgreSQL`, `SQLite`, and `SQL Server`, it enables developers to work with the database technology that best suits their needs.

Using the query builder provides a number of benefits, one of which is access to an expressive syntax. It offers a flexible interface that makes it possible for developers to chain methods and construct queries in a way that is both clear and easy to understand. Consider, for instance, the following snippet of code, which, when applied to a "users" database, retrieves all of the users:

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

In this demonstration, we begin by identifying the table (users) that will serve as the target of our query. After that, in order to filter and sort the data, we chain together the select, where, and orderBy methods. In the final step, we receive the results by executing the query and then calling the get method.

The query builder also includes a broad variety of methods to make complicated inquiries, such as joins, subqueries, aggregate functions, and conditional clauses. Using these methods, I can retrieve the information I need from the database. It makes working with databases and writing code that is easy to maintain easier by providing a syntax that is user-friendly and straightforward for performing these tasks.

The fact that the query builder has built-in support for parameter binding is another one of its many benefits. This function automatically cleans up user input in order to assist avoid SQL injection attacks, which can be very damaging. I can avoid immediately concatenating values into the query by making use of placeholders and then passing the values that correspond to those placeholders as distinct query parameters. The query builder will take care of appropriately escaping the values for I, ensuring that my database operations are carried out in a secure and trustworthy manner.

In addition, the query builder has a seamless integration with other elements of Laravel, such as the Eloquent ORM (Object-Relational Mapping) and the Blade templating engine. Within the Laravel ecosystem, this enables developers to utilize the power of these technologies in tandem with the query builder, which enables efficient and flexible database interactions.

In general, the query builder that comes packaged with Laravel offers a mechanism to interface with databases that is clear, easily readable, and safe. It makes working with databases easier and more pleasurable for developers by hiding the complexity of raw SQL behind a higher degree of abstraction and by abstracting away the complexities of SQL itself.

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.

Answer -

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

print_r($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?

Answer -

The `distinct()` method in Laravel's query builder is used to retrieve unique rows from a database table. It ensures that the result set contains only distinct (unique) values, eliminating any duplicate rows that may exist based on the specified columns.

When used in conjunction with the `select()` method, the `distinct()` method allows I to apply the distinct operation to a specific set of columns. By default, if no columns are provided to the `distinct()` method, it considers all columns in the select() method for determining uniqueness.

Here's an example to illustrate the usage of `distinct()` in conjunction with the `select()` method:

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

The `distinct()` method is useful in scenarios where I want to eliminate duplicate rows from the result set, such as when generating reports, calculating statistics, or performing data analysis. It helps ensure data accuracy and prevents duplicate information from skewing my 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.

Answer -

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

if ($posts) {
    print_r($posts->description);
} else {
    echo "No post found.";
}
```

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.

Answer -

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

print_r($posts);
```

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

Answer -

Both the `first()` and `find()` methods can be used in the query builder of Laravel to retrieve a single record from the database. However, the way in which they identify the record to retrieve is different between the two ways.

In most cases, the `first()` method is utilized to acquire the first entry in the set that satisfies the requirements that have been stated. It gives back the first row that satisfies the query requirements and then applies the order that was supplied, if there was one. The following is an illustration of how the `first()` function may be used to fetch the first record from the "users" table:

```
$user = DB::table('users')
            ->where('age', '>', 18)
            ->orderBy('created_at', 'asc')
            ->first();
```

On the other hand, the `find()` method is used to retrieve a record based on its primary `key value`. It assumes that the primary key column in the table is named "id" by default. Here's an example of using `find()` to retrieve a specific user by their ID:

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

The key difference between `first()` and `find()` lies in how they identify the record to retrieve. `first()` allows I to specify conditions and order to retrieve the first matching record, while `find()` retrieves a record based on its primary key value.

It's important to note that both methods return the record as an object, so I can access its properties using object notation (e.g., `$user->name`). Additionally, if no matching record is found, both methods return null.

By understanding the distinction between `first()` and `find()`, I can choose the appropriate method based on my specific requirements when retrieving single records from the database in Laravel.

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.

Answer -

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

```php
print_r($posts);
```

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.

Answer -

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

print_r($result);
```

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.

Answer -

```php
$posts = DB::table('posts')
    ->where('id', 2)
    ->update([
        'excerpt' => 'Laravel 10',
        'description' => 'Laravel 10 is the latest version of Laravel, the popular PHP frame
    ]);

$affectedRows = $posts->rowCount();
echo "Number of affected rows: $affectedRows";
```

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.

Answer -

```php
DB::table('posts')
```

```
    ->where('id', 3)
    ->delete();

$affectedRows = DB::affectedRows();
echo $affectedRows;
```

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.

Answer -

- `count()` - This method returns the number of records that match the query conditions.

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

- `sum()` - This method calculates the sum of the values in a specified column.

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

- `avg()` - This method calculates the average of the values in a specified column.

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

- `max()` - This method retrieves the maximum value from a specified column.

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

- `min()` - This method retrieves the minimum value from a specified column.

```
$lowestAge = DB::table('users')->min('age');
```

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

Answer -

The whereNot() method in Laravel's query builder is used to add a "where not" clause to a query. This clause is used to filter out rows from the results that match the specified condition. The whereNot() method takes a single argument, which is a closure that contains the condition to be checked. The closure is passed an instance of the Query Builder, which can be used to access the database and build the query.

Here is an example of how the whereNot() method can be used:

```
// Get all users who are not active
$users = DB::table('users')
    ->whereNot('active', 1)
    ->get();
```

This query will return all rows from the users table where the active column is not equal to 1.

The whereNot() method can also be used to chain multiple conditions together. For example, the following query will return all users who are not active and who have not created any posts:

```
$users = DB::table('users')
    ->whereNot('active', 1)
    ->whereNot('posts_count', 0)
    ->get();
```

The whereNot() method is a powerful tool that can be used to filter out rows from the results of a query. It can be used to create complex queries that can be used to retrieve the exact data that is needed.

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?

Answer -

- `exists()` - This method checks if there are any records that match the query conditions and returns a boolean value indicating whether the records exist or not.

```
if (DB::table('users')->where('name', 'John')->exists()) {
    // Records with name 'John' exist
} else {
    // No records with name 'John' exist
}
```

- `doesntExist()` - This method checks if there are no records that match the query conditions and returns a boolean value indicating whether the records don't exist.

```
if (DB::table('users')->where('name', 'John')->doesntExist()) {
    // No records with name 'John' exist
} else {
    // Records with name 'John' exist
}
```

By using exists() and doesntExist() methods, i can easily validate the existence of records in my database tables and make decisions based on the results.

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.

Answer -

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

foreach ($posts as $post) {
    echo $post->title . ' ' . $post->min_to_read . PHP_EOL;
}
```

This code will first create a new `Post` object and then use the `whereBetween` method to filter the results to only include posts where the `min_to_read` column is between 1 and 5. The results will then be stored in the `$posts` variable. Finally, the `$posts` variable will be looped through and each post's title and `min_to_read` value will be printed.

```
Title Min To Read
------- --------
Post 1 1
Post 2 2
Post 3 3
Post 4 4
Post 5 5
```

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.

Answer -

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

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

echo $affectedRows . ' rows affected';
```

This code will first increment the `min_to_read` column value of the record with the `id` of 3 by 1. Then, it will count the number of rows that were affected by the update. Finally, it will print the number of affected rows.