

1. Explanation of Laravel's Query Builder:

Laravel's query builder provides a simple and elegant way to interact with databases. It allows you to build database queries using a fluent, chainable interface, which makes it easier to construct complex queries while keeping your code readable and maintainable. The query builder abstracts the underlying database system and provides a unified API to interact with multiple database engines supported by Laravel, such as MySQL, PostgreSQL, SQLite, and others. It eliminates the need to write raw SQL queries by hand, reducing the risk of SQL injection and making your code more database-agnostic.

With the query builder, you can perform various database operations like selecting, inserting, updating, and deleting records. It provides methods to specify conditions, order results, join tables, aggregate data, and more. The query builder's methods are expressive and closely resemble the SQL syntax, making it intuitive for developers familiar with SQL.

Overall, Laravel's query builder simplifies database interactions by providing an elegant and readable API, improving security, and ensuring compatibility with different database systems.

2. Code to retrieve columns from the "posts" table:

```
1 use Illuminate\Support\Facades\DB;  
2  
3 $posts = DB::table('posts')->select('excerpt', 'description')->get();  
4  
5 print_r($posts);|  
6
```

3. Purpose of the distinct() method:

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

The distinct() method is often used in conjunction with the select() method. When you call select() and pass column names, the query builder retrieves all the rows from the database that match the conditions specified in the query. However, if you want to retrieve only the unique values of specific columns, you can chain the distinct() method before the select() method.

For example:

```
1 use Illuminate\Support\Facades\DB;  
2  
3 $posts = DB::table('posts')  
4     ->distinct()  
5     ->select('title')  
6     ->get();  
7
```

In this code, the distinct() method is used before select('title'), ensuring that only unique titles are retrieved from the "posts" table. This can be useful when you want to fetch a list of distinct values from a column.

4. Code to retrieve the first record with id 2 from the "posts" table:

```
1 use Illuminate\Support\Facades\DB;  
2  
3 $posts = DB::table('posts')->where('id', 2)->first();  
4  
5 echo $posts->description;  
6
```

This code uses Laravel's query builder to retrieve the first record from the "posts" table where the "id" is 2. The where('id', 2) method specifies the condition for the query. The first() method retrieves the first matching record. The result is stored in the \$posts variable. Finally, echo \$posts->description prints the "description" column of the \$posts variable.

5. Code to retrieve the "description" column from the "posts" table where the "id" is 2:

```
1 use Illuminate\Support\Facades\DB;  
2  
3 $posts = DB::table('posts')->where('id', 2)->value('description');  
4  
5 echo $posts;  
6
```

This code uses Laravel's query builder to retrieve the "description" column from the "posts" table where the "id" is 2. The where('id', 2) method specifies the condition for the query. The value('description') method retrieves the value of the "description" column directly without the need for an array. The result is stored in the \$posts variable. Finally, echo \$posts prints the value of the \$posts variable.

6. Difference between the first() and find() methods:

The first() and find() methods in Laravel's query builder are used to retrieve single records from the database, but they have some differences in their usage.

- The `first()` method retrieves the first record matching the query conditions. It returns a single object representing the first row found. It is commonly used when you want to fetch the first result from a query result set.

```
1 use Illuminate\Support\Facades\DB;  
2  
3 $post = DB::table('posts')->where('category', 'News')->first();  
4 |
```

In this example, the `first()` method retrieves the first record from the "posts" table where the "category" is 'News' and stores it in the `$post` variable.

- The `find()` method is used to retrieve a record by its primary key value. It expects the primary key value as an argument and returns the matching record as a single object. It is commonly used when you know the primary key value and want to retrieve a specific record.

```
1 use Illuminate\Support\Facades\DB;  
2  
3 $post = DB::table('posts')->find(2);  
4 |
```

In this example, the `find(2)` method retrieves the record from the "posts" table with the primary key value of 2 and stores it in the `$post` variable.

Both `first()` and `find()` methods can be useful in different scenarios depending on the requirement to retrieve a single record.

7. Code to retrieve the "title" column from the "posts" table:

```
1 use Illuminate\Support\Facades\DB;  
2  
3 $posts = DB::table('posts')->pluck('title');  
4  
5 print_r($posts);  
6 |
```

This code uses Laravel's query builder to retrieve the "title" column from the "posts" table. The `pluck('title')` method retrieves only the values of the "title" column and returns them as an

array. The result is stored in the \$posts variable. Finally, print_r(\$posts) prints the contents of the \$posts variable

8. Code to insert a new record into the "posts" table:

```
1  use Illuminate\Support\Facades\DB;  
2  
3  $result = DB::table('posts')->insert([  
4      'title'        => 'X',  
5      'slug'         => 'X',  
6      'excerpt'      => 'excerpt',  
7      'description'  => 'description',  
8      'is_published' => true,  
9      'min_to_read'  => 2,  
10 ]);  
11  
12 print_r($result);  
13
```

This code uses Laravel's query builder to insert a new record into the "posts" table. The insert() method is called on the query builder instance with an array of column-value pairs representing the data to be inserted. The result of the insert operation is stored in the \$result variable. Finally, print_r(\$result) prints the result, which will be true if the insert operation was successful.

9. Code to update the "excerpt" and "description" columns of a record:

```
1  use Illuminate\Support\Facades\DB;  
2  
3  $affectedRows = DB::table('posts')  
4      ->where('id', 2)  
5      ->update([  
6          'excerpt'    => 'Laravel 10',  
7          'description' => 'Laravel 10',  
8      ]);  
9  
10 echo $affectedRows;  
11
```

This code uses Laravel's query builder to update the "excerpt" and "description" columns of the record with the "id" of 2 in the "posts" table. The where('id', 2) method specifies the condition to identify the record. The update() method is called on the query builder instance with an array of column-value pairs representing the new values. The number of affected rows is stored in the \$affectedRows variable. Finally, echo \$affectedRows prints the number of affected rows.

10. Code to delete a record from the "posts" table:

```
1  use Illuminate\Support\Facades\DB;  
2  
3  $affectedRows = DB::table('posts')->where('id', 3)->delete();  
4  
5  echo $affectedRows;  
6  |
```

This code uses Laravel's query builder to delete the record with the "id" of 3 from the "posts" table. The where('id', 3) method specifies the condition to identify the record. The delete() method is called on the query builder instance to perform the deletion. The number of affected rows is stored in the \$affectedRows variable. Finally, echo \$affectedRows prints the number of affected rows.

11. Explanation of aggregate methods in Laravel's query builder:

Laravel's query builder provides several aggregate methods that allow you to perform calculations on a set of values in a database column. Here are the commonly used aggregate methods:

count(): Calculates the number of rows matching the query conditions.

```
1  use Illuminate\Support\Facades\DB;  
2  
3  $count = DB::table('posts')->count();  
4  |
```

sum(): Calculates the sum of the values in a column.

```
1 use Illuminate\Support\Facades\DB;  
2  
3 $total = DB::table('orders')->sum('amount');  
4 |
```

avg(): Calculates the average value of a column.

```
1 use Illuminate\Support\Facades\DB;  
2  
3 $average = DB::table('products')->avg('price');  
4 |
```

max(): Retrieves the maximum value from a column.

```
1 use Illuminate\Support\Facades\DB;  
2  
3 $maxPrice = DB::table('products')->max('price');  
4 |
```

min(): Retrieves the minimum value from a column.

```
1 use Illuminate\Support\Facades\DB;  
2  
3 $minPrice = DB::table('products')->min('price');  
4 |
```

These aggregate methods are used to perform calculations on columns and retrieve aggregated results from the database.

12. Usage of the whereNot() method:

The whereNot() method in Laravel's query builder is used to add a "not equals" condition to a query. It specifies that the column value should not be equal to the provided value. Here's an example of how the whereNot() method is used:

```
1 use Illuminate\Support\Facades\DB;  
2  
3 $posts = DB::table('posts')->whereNot('status', 'published')->get();  
4
```

In this code, the `whereNot('status', 'published')` method adds a condition to the query that retrieves all the posts where the "status" column is not equal to 'published'. The `get()` method executes the query and retrieves the results, which are stored in the `$posts` variable. The `whereNot()` method is useful when you want to exclude specific values from the result set based on a column's value.

13. Difference between exists() and doesntExist() methods:

The `exists()` and `doesntExist()` methods in Laravel's query builder are used to check the existence of records based on a query.

The `exists()` method is used to check if records exist based on a query. It returns true if the query returns any results and false otherwise.

```
1 use Illuminate\Support\Facades\DB;  
2  
3 $exists = DB::table('users')->where('email', 'john@example.com')->exists();  
4
```

In this example, the `exists()` method checks if there are any records in the "users" table with the email 'john@example.com'. It returns true if such records exist and false otherwise.

The `doesntExist()` method is the negation of `exists()`. It returns true if the query does not return any results and false otherwise.

```
1 use Illuminate\Support\Facades\DB;  
2  
3 $doesntExist = DB::table('users')->where('email', 'john@example.com')->doesntExist();  
4
```

In this example, the `doesntExist()` method checks if there are no records in the "users" table with the email 'john@example.com'. It returns true if no such records exist and false otherwise.

These methods are useful when you want to perform conditional checks based on the existence or non-existence of records in the database.

14. Code to retrieve records with min_to_read between 1 and 5:

```
1 use Illuminate\Support\Facades\DB;  
2  
3 $posts = DB::table('posts')->whereBetween('min_to_read', [1, 5])->get();  
4  
5 print_r($posts);  
6 |
```

This code uses Laravel's query builder to retrieve records from the "posts" table where the "min_to_read" column is between 1 and 5. The whereBetween('min_to_read', [1, 5]) method adds a condition to the query to retrieve records where the "min_to_read" value is within the specified range. The get() method executes the query and retrieves the results, storing them in the \$posts variable. Finally, print_r(\$posts) prints the contents of the \$posts variable.

15. Code to increment the "min_to_read" column by 1:

```
1 use Illuminate\Support\Facades\DB;  
2  
3 $affectedRows = DB::table('posts')->where('id', 3)->increment('min_to_read');  
4  
5 echo $affectedRows;  
6 |
```

This code uses Laravel's query builder to increment the "min_to_read" column value of the record with the "id" of 3 in the "posts" table by 1. The where('id', 3) method specifies the condition to identify the record. The increment('min_to_read') method increments the value of the "min_to_read" column by 1. The number of affected rows is stored in the \$affectedRows variable. Finally, echo \$affectedRows prints the number of affected rows.