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 feature provided by the Laravel framework, which offers a convenient and expressive way to interact with databases. It allows you to build database queries using a fluent, chainable interface, making it easier to fetch, insert, update, and delete records in your database.

The query builder provides a higher-level abstraction over raw SQL queries, allowing you to write database queries using PHP code instead. It abstracts away the complexities of writing raw SQL statements while providing a more intuitive and readable syntax. This approach is known as the "fluent interface" design pattern, where method calls are chained together to form the query.

Here's an example of a basic query using Laravel's query builder:

Another advantage of the query builder is its support for parameter binding. It automatically handles the binding of parameters, preventing SQL injection vulnerabilities and ensuring proper data sanitization.

By utilizing the query builder, you can write database queries in a concise and readable manner, improving the maintainability and readability of your code. It abstracts away the low-level details of interacting with the database, allowing you to focus more on your application's logic and less on SQL syntax.

Overall, Laravel's query builder provides a powerful and elegant way to interact with databases, simplifying database operations and enhancing developer productivity.

==========

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.

Here's the code to retrieve the "excerpt" and "description" columns from the "posts" table using Laravel's query builder and store the result in the `$posts` variable:

```php
public function q2(){
        $posts = DB::table('posts')
        ->select('excerpt', 'description')
        ->get();
        return Response()->json([
            'success'=>true,
            'data'=>$posts
        ], 201);
    }
```

In this code, we're using the `DB::table('posts')` method to specify the "posts" table as the target of our query. Then, we use the `select()` method to specify the columns we want to retrieve, which are "excerpt" and "description" in this case.

Finally, we use the `get()` method to execute the query and retrieve the results. The result will be stored in the `$posts` variable. You can then use `return $posts` to print the contents of the `$posts` variable, which will display the retrieved records.

==========

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 specific column or set of columns in the result set. It ensures that duplicate values are eliminated, and only distinct values are returned.

When used in conjunction with the `select()` method, the `distinct()` method modifies the behavior of the `select()` method to retrieve distinct values for the specified columns.

Here's an example to illustrate its usage:

```php
public function q3(){
        $uniqueTitle = DB::table('posts')
                ->select('title')
                ->distinct()
                ->get();

            return response()->json([
                    'success'=>true,
                    'message'=>$uniqueTitle
                ], 201);
    }
```

The `distinct()` method is useful when you want to eliminate duplicate values from the result set and work with unique values only. It can be handy when dealing with scenarios such as generating unique lists, performing calculations on distinct values, or eliminating redundant data from your queries.

==========

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.

Here's 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, and print the "description" column:

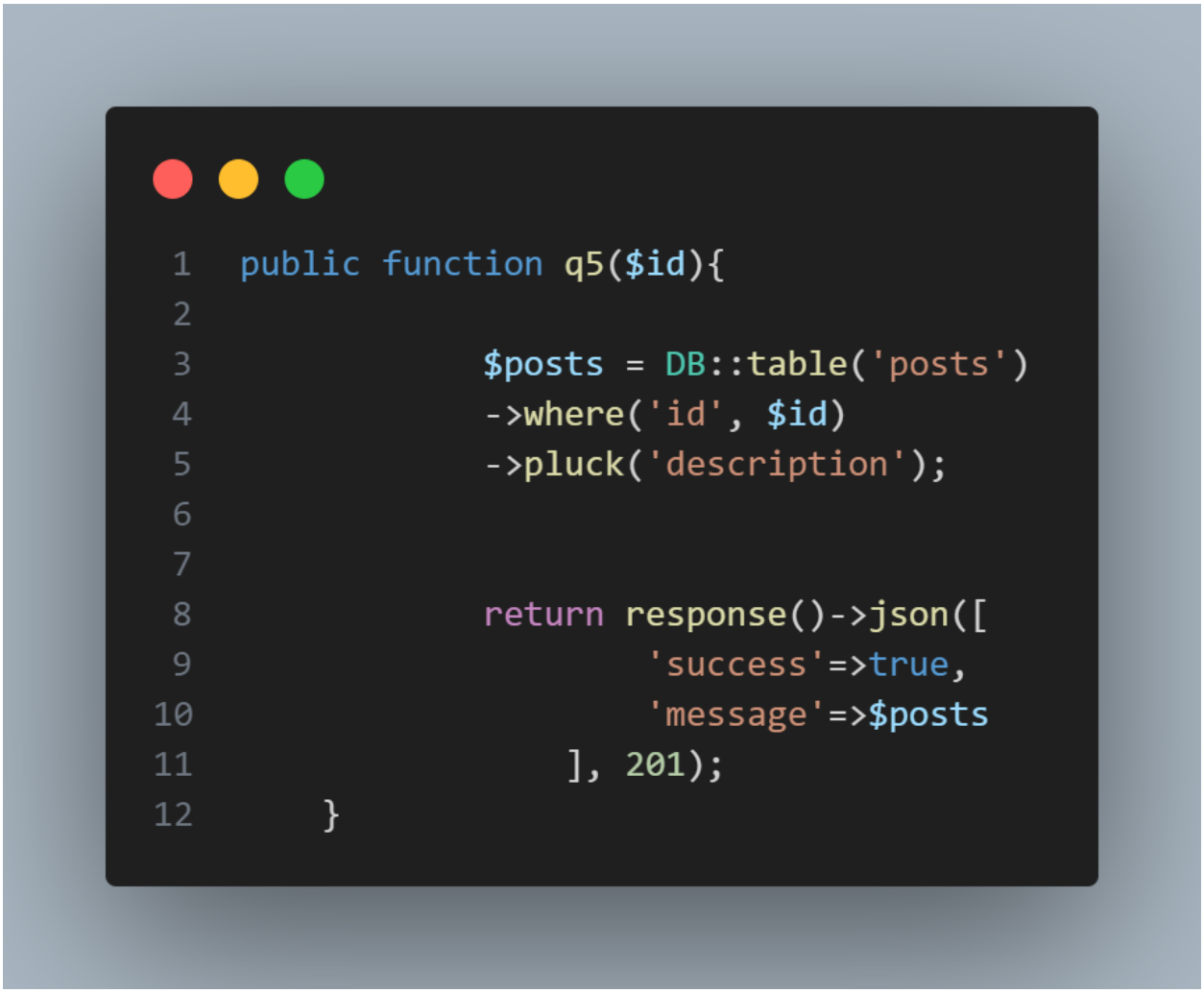
```php
public function q4($id){

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

            return response()->json([
                    'success'=>true,
                    'message'=>$posts->description??'No Post Found.'
                ], 201);
    }
```

In this code, we're using the `DB::table('posts')` method to specify the "posts" table as the target of our query. We then use the `where()` method to filter the records and retrieve the first record where the "id" column is equal to 2.

==========

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.

Here's 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 `$posts` variable, and print the `$posts` variable:



```php
public function q5($id){

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


            return response()->json([
                    'success'=>true,
                    'message'=>$posts
                ], 201);
    }
```

In this code, we're using the `DB::table('posts')` method to specify the "posts" table as the target of our query. We then use the `where()` method to filter the records and retrieve the record where the "id" column is equal to 2.

The `pluck('description')` method is used to retrieve the value of the "description" column directly. It returns an array containing only the values of the specified column.

The result is stored in the `$posts` variable, which will contain the value of the "description" column for the record with an "id" of 2.

Finally, we use `return($posts)` to print the contents of the `$posts` variable, which will display the retrieved "description" value.

==========

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

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

The `first()` method is used to retrieve the first record that matches the query criteria. It is typically used when you want to retrieve the first result based on a specific ordering or filtering condition. Here's an example:

```php
public function q6First(){

        $firstPost = DB::table('posts')
        ->where('is_published', 1)
        ->orderBy('created_at')
        ->first();



            return response()->json([
                    'success'=>true,
                    'message'=>$firstPost
                ], 201);
    }
```

In this example, we're retrieving the first post from the "posts" table that has a status of "published." We order the results by the "created_at" column, and `first()` returns the first matching record based on this ordering.

On the other hand, the `find()` method is used to retrieve a record by its primary key. It assumes that the primary key of the table is named "id" by default, but you can customize it if needed. Here's an example:

```php
public function q6Find($id){

        $post = DB::table('posts')->find($id);
            return response()->json([
                    'success'=>true,
                    'message'=>$post
                ], 201);
    }
```

In this example, we're retrieving a post from the "posts" table with an "id" of 2. The `find(2)` method searches for a record with a primary key value of 2 and returns it.

To summarize, the main difference between `first()` and `find()` is that `first()` retrieves the first record based on ordering or filtering conditions, while `find()` retrieves a record by its primary key value.

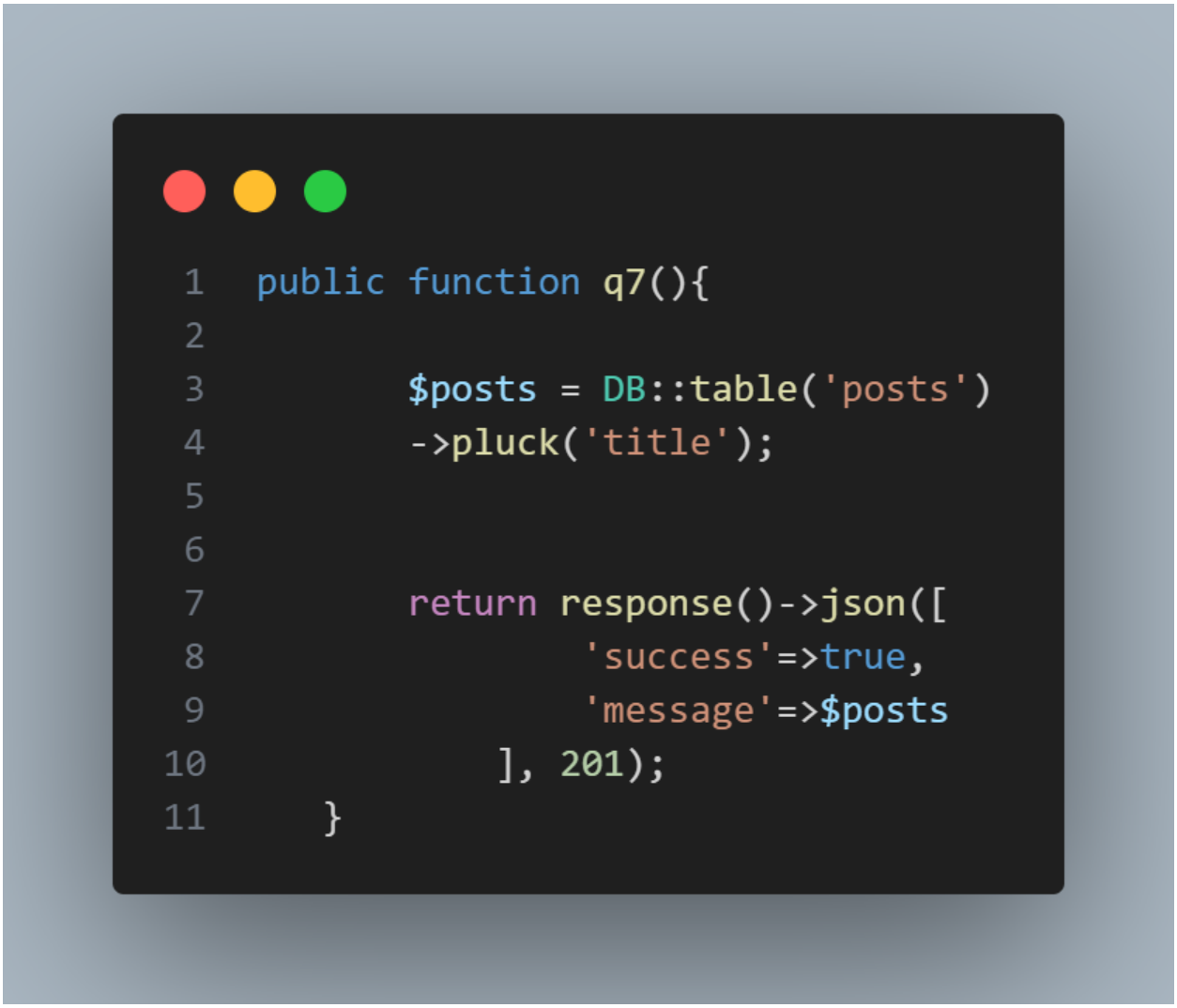
It's important to note that both methods return a single record as an object if a matching record is found. If no record is found, `first()` returns `null`, while `find()` returns `null` or an empty model instance, depending on the Laravel version.

Additionally, the `find()` method provides a convenient way to retrieve records by their primary key, making it useful for scenarios where you have the specific primary key value at hand.

==========

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.

Here's the code to retrieve the "title" column from the "posts" table using Laravel's query builder, store the result in the `$posts` variable, and print the `$posts` variable:



```php
public function q7(){

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


        return response()->json([
                'success'=>true,
                'message'=>$posts
            ], 201);
    }
```

In this code, we're using the `DB::table('posts')` method to specify the "posts" table as the target of our query.

The `pluck('title')` method is used to retrieve the values of the "title" column directly. It returns an array containing only the values of the specified column.

The result is stored in the `$posts` variable, which will contain an array of the "title" values from the "posts" table.

Finally, we use `return($posts)` to print the contents of the `$posts` variable, which will display the retrieved "title" values.

==========

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.

Here's the code to insert a new record into the "posts" table using Laravel's query builder, setting the specified column values and printing the result of the insert operation:

```php
public function q8(Request $request){

        $post = DB::table('posts')
                    ->insert([
                        'user_id'=>$request->input('user_id'),
                        'title'=>$request->input('title'),
                        'slug'=>$request->input('slug'),
                        'excerpt'=>$request->input('excerpt'),
                        'description'=>$request->input('description'),
                        'is_published'=>$request->input('is_published'),
                        'min_to_read'=>$request->input('min_to_read'),
                    ]);

                if(!$post) {
                    return Response()->json([
                        'success'=>false,
                        'message'=>'Something Error. Try again'
                    ]);
                } else {
                    return response()->json([
                        'success'=>true,
                        'message'=>$post
                    ], 201);
                }


    }
```

In this code, we're using the `DB::table('posts')` method to specify the "posts" table as the target of our insert operation.

The `insert()` method is used to insert a new record into the table. We pass an associative array as an argument, where the keys correspond to the column names and the values represent the data we want to insert.

In this example, we're setting the "title" and "slug" columns to 'X', the "excerpt" column to 'excerpt', the "description" column to 'description', the "is_published" column to true, and the "min_to_read" column to 2.

The `insert()` method returns a boolean value indicating whether the insert operation was successful. We then use a conditional statement to print the appropriate message based on the 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.

Here's 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. We'll set the new values to 'Laravel 10' and print the number of affected rows:

```php
public function q9($id) {
        $affectedRows = DB::table('posts')
                ->where('id', $id)
                ->update([
                    'excerpt' => 'Laravel 10',
                    'description' => 'Laravel 10'
                ]);

            return response()->json([
                'success'=>true,
                'message'=>'number of affected rows '.$affectedRows
            ], 201);

    }
```

In this code, we're using the `DB::table('posts')` method to specify the "posts" table as the target of our update operation.

The `where('id', 2)` method is used to filter the records and select the record with an "id" of 2 for updating.

The `update()` method is used to perform the update operation on the selected record. We pass an associative array as an argument, where the keys correspond to the column names we want to update, and the values represent the new data we want to set.

In this example, we're updating the "excerpt" and "description" columns to 'Laravel 10'.

The `update()` method returns the number of affected rows as an integer, representing the number of records that were successfully updated. We then echo the number of affected rows using the `$affectedRows` variable.

==========

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.

Here's the code to delete the record with the "id" of 3 from the "posts" table using Laravel's query builder and print the number of affected rows:

```php
public function q10($id) {
        $affectedRows = DB::table('posts')
                ->where('id', $id)
                ->delete();


            return response()->json([
                'success'=>true,
                'message'=>'number of affected rows '.$affectedRows
            ], 201);

    }
```

In this code, we're using the `DB::table('posts')` method to specify the "posts" table as the target of our delete operation.

The `where('id', 3)` method is used to filter the records and select the record with an "id" of 3 for deletion.

The `delete()` method is used to perform the delete operation on the selected record. It removes the record from the table and returns the number of affected rows as an integer.

We store the number of affected rows in the `$affectedRows` variable and then echo the number of affected rows using `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.

In Laravel's query builder, aggregate methods such as `count()`, `sum()`, `avg()`, `max()`, and `min()` are used to perform calculations on a set of database records. These methods allow you to retrieve aggregated results, such as the count of records, the sum of a specific column, the average value of a column, the maximum value in a column, and the minimum value in a column. Here's an explanation and example of each:

1. `count()`: The `count()` method is used to retrieve the number of records in a table or the number of records that match a specific condition. It returns the count as an integer value.

2. `sum()`: The `sum()` method is used to calculate the sum of a specific column's values. It returns the sum as a numeric value.

3. `avg()`: The `avg()` method is used to calculate the average (mean) value of a specific column's values. It returns the average as a numeric value.

4. `max()`: The `max()` method is used to retrieve the maximum value of a specific column. It returns the maximum value as a numeric or string value, depending on the column type.

5. `min()`: The `min()` method is used to retrieve the minimum value of a specific column. It returns the minimum value as a numeric or string value, depending on the column type.

```php
public function q11() {
        $posts['count'] = DB::table('posts')->count();
        $posts ['sum']= DB::table('posts')->sum('min_to_read');
        $posts ['avg'] = DB::table('posts')->avg('min_to_read');
        $posts ['max']= DB::table('posts')->max('min_to_read');
        $posts ['min']= DB::table('posts')->min('min_to_read');


            return response()->json([
                'success'=>true,
                'data'=>$posts
            ], 201);

    }
```

These aggregate methods are useful when you need to perform calculations on a set of records or retrieve statistical information from the database. They allow you to easily retrieve aggregated results without having to manipulate the data manually.

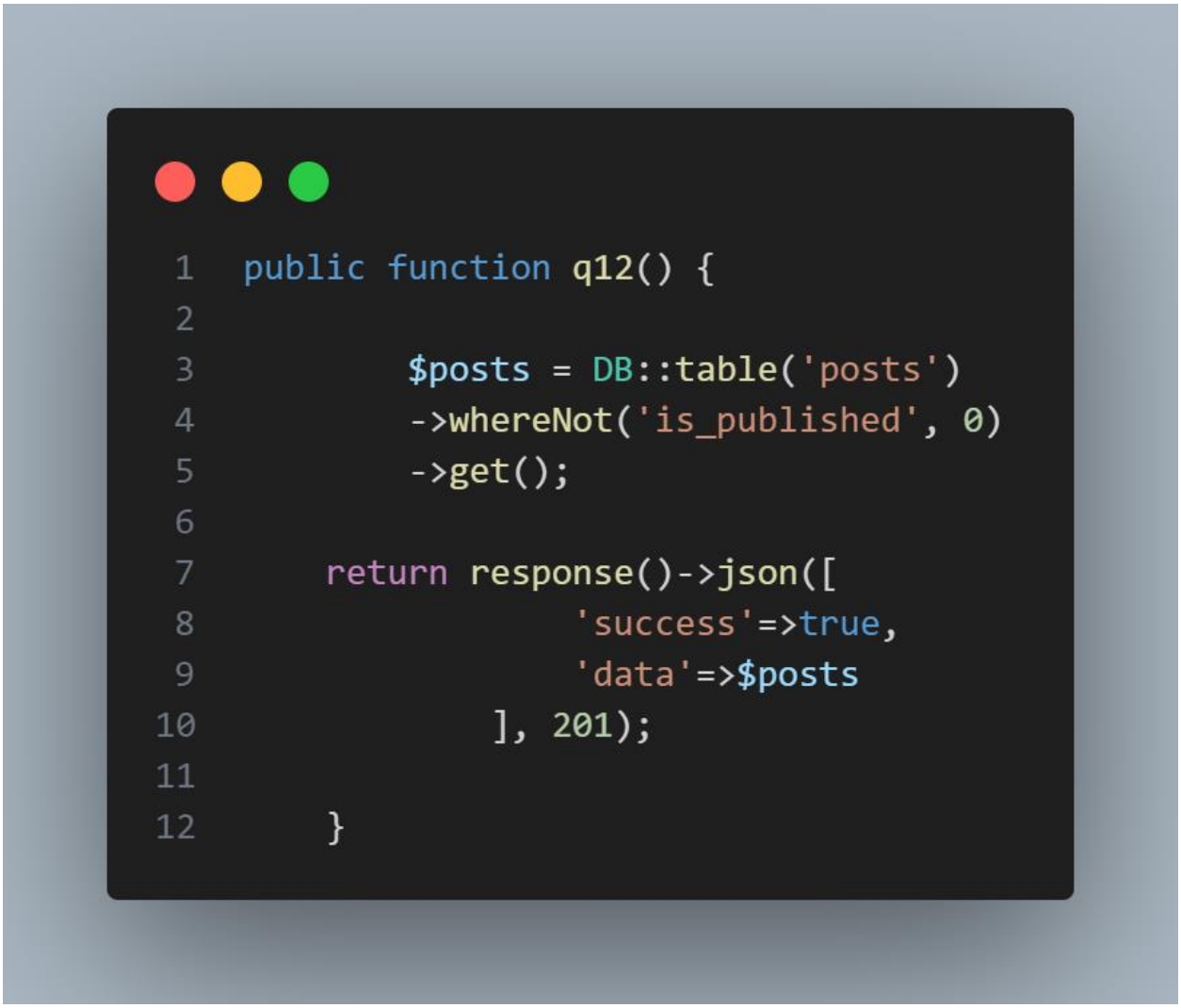
==========

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

In Laravel's query builder, the `whereNot()` method is used to add a "not equal" condition to a query. It allows you to specify a column and a value that the column should not equal in the query.

The `whereNot()` method can be used in conjunction with other query builder methods like `select()`, `where()`, or `orWhere()` to further refine the query.

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



```php
public function q12() {

        $posts = DB::table('posts')
        ->whereNot('is_published', 0)
        ->get();

    return response()->json([
                'success'=>true,
                'data'=>$posts
            ], 201);

    }
```

In this example, we're querying the "users" table and using the `whereNot('is_published', 1)` method to filter the records. This query will retrieve all the users whose "status" column is not equal to 1.

The `whereNot()` method takes two arguments: the column name and the value to compare against. It adds a "not equal" condition to the query, filtering out the records that match the specified column and value.

You can chain additional methods like `get()`, `orderBy()`, or `limit()` to further modify the query according to your requirements.

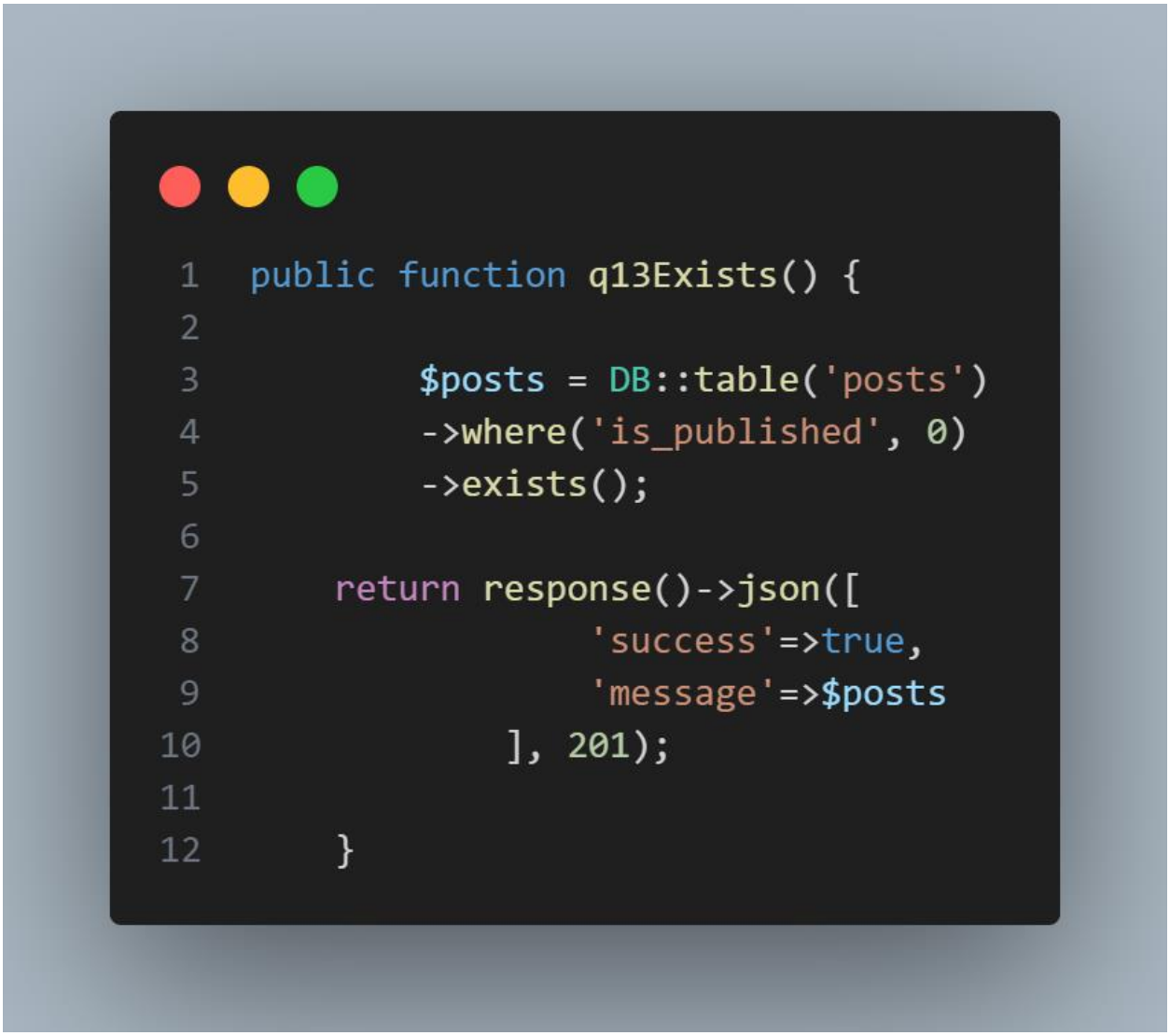
It's important to note that the `whereNot()` method is different from the `where()` method with a "not equal" condition. The `whereNot()` method specifically adds a "not equal" condition, while the `where()` method with a "not equal" condition can be achieved by using the `<>` operator.

==========

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?

In Laravel's query builder, the `exists()` and `doesntExist()` methods are used to check the existence of records in a table. However, they differ in their approach and the condition they check.

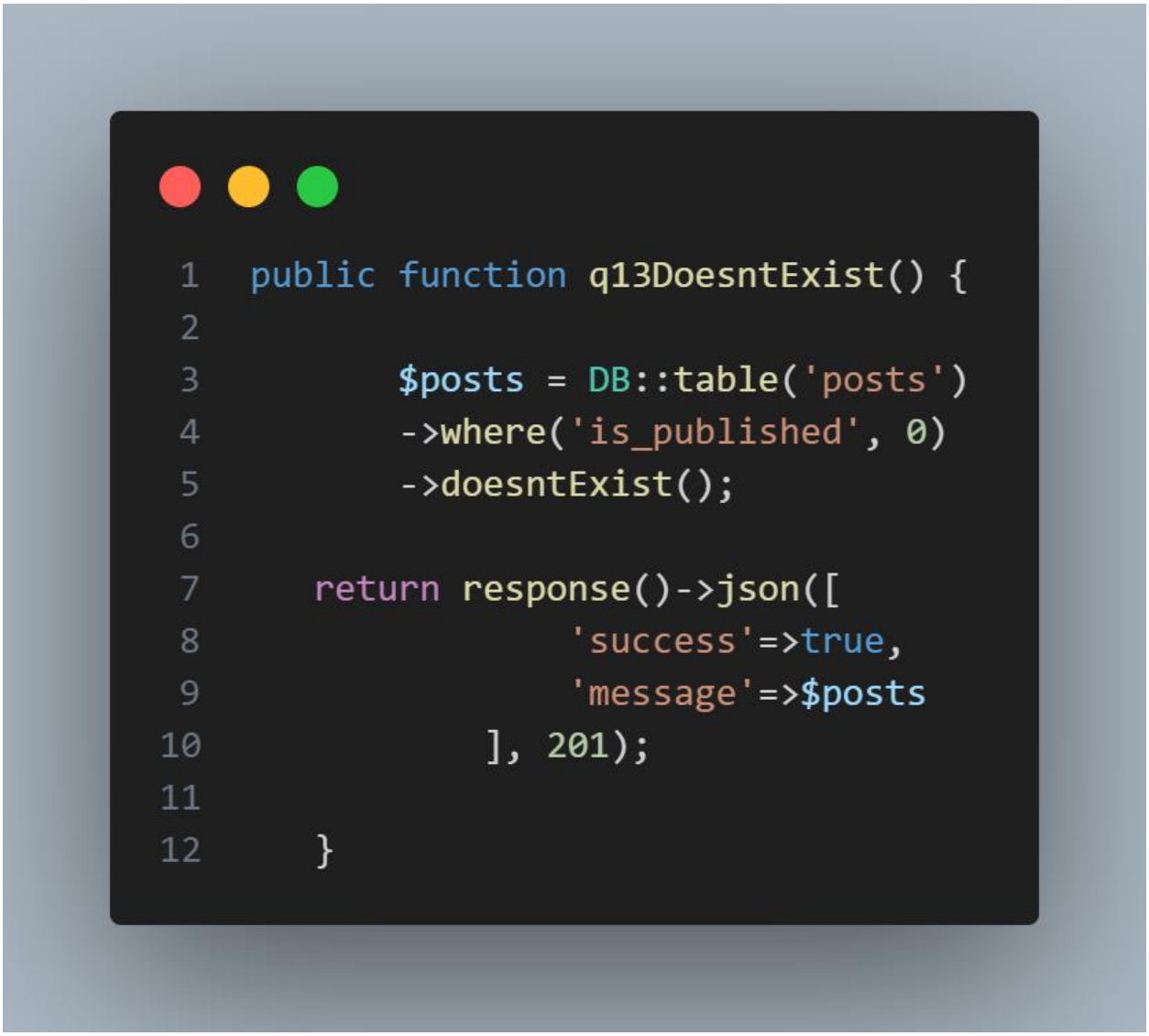
The `exists()` method is used to check if any records exist in the table that match a given condition. It returns `true` if at least one record exists, and `false` otherwise. Here's an example:



```
1  public function q13Exists() {
2
3          $posts = DB::table('posts')
4          ->where('is_published', 0)
5          ->exists();
6
7      return response()->json([
8                  'success'=>true,
9                  'message'=>$posts
10             ], 201);
11
12     }
```

In this example, we're using the `exists()` method to check if there are any active users in the "users" table. The `where('is_published', 1)` condition filters the records based on the "status" column. If at least one record matches the condition, `exists()` will return `true`, indicating that users exist.

On the other hand, the `doesntExist()` method is used to check if no records exist in the table that match a given condition. It returns `true` if no records are found, and `false` if there is at least one matching record. Here's an example:



```php
public function q13DoesntExist() {

        $posts = DB::table('posts')
        ->where('is_published', 0)
        ->doesntExist();

    return response()->json([
                'success'=>true,
                'message'=>$posts
            ], 201);

    }
```

In this example, we're using the `doesntExist()` method to check if there are no inactive users in the "users" table. The `where('is_published', 'inactive')` condition filters the records based on the "status" column. If no record matches the condition, `doesntExist()` will return `true`, indicating that no inactive users exist.

Both methods provide a convenient way to check the existence or non-existence of records based on specific conditions, allowing you to handle different scenarios in your application based on the presence or absence of records.

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.

Here's 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. We'll store the result in the `$posts` variable and then print the `$posts` variable:

```php
public function q14() {

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


    return response()->json([
                'success'=>true,
                'data'=>$posts
            ], 201);

    }
```

In this code, we're using the `DB::table('posts')` method to specify the "posts" table as the target of our query.

The `whereBetween('min_to_read', [1, 5])` method is used to filter the records based on the "min_to_read" column, selecting only the records where the value falls between 1 and 5 (inclusive).

The `get()` method is used to retrieve the records that match the specified conditions. It returns a collection of objects, each representing a row from the table.

We store the result in the `$posts` variable, and then use `return($posts)` to print the contents of the `$posts` variable, which will display the retrieved records.

==========

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.

Here's 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. We'll also print the number of affected rows:

```php
public function q15($id) {

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

    return response()->json([
                'success'=>true,
                'message'=>'number of affected rows '.$affectedRows
            ], 201);

    }
```

In this code, we're using the `DB::table('posts')` method to specify the "posts" table as the target of our update operation.

The `where('id', 3)` method is used to filter the records and select the record with an "id" of 3 for the update.

The `increment('min_to_read', 1)` method is used to increment the value of the "min_to_read" column by 1 for the selected record. The first argument specifies the column to increment, and the second argument specifies the increment value.

The `increment()` method returns the number of affected rows as an integer, representing the number of records that were successfully updated. We then echo the number of affected rows using `echo "Number of affected rows: " . $affectedRows;`.