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

Answer: Laravel query builder is a feature provided by the Laravel framework that allows developers to interact with databases using a fluent and intuitive syntax. It provides a simple and elegant way to perform database operations without directly writing SQL queries.

Through query builder, we can write database queries using the query builder syntax, and Laravel will take care of generating the appropriate SQL statements for the specific database we're using.

Here are some key aspects of Laravel's query builder that make it simple and elegant:

1. Fluent method chaining: The query builder uses a fluent interface, which means you can chain methods together to build your queries in a highly readable and expressive way.

2. Automatic parameter binding: Laravel's query builder automatically handles parameter binding, which helps protect against SQL injection attacks.

3. Eloquent ORM integration: Laravel's query builder seamlessly integrates with its powerful ORM (Object-Relational Mapping) called Eloquent. Eloquent provides an active record implementation for working with database records, and the query builder is used behind the scenes to build and execute the queries needed for data retrieval, insertion, updating, and deletion.

In summary, Laravel's query builder simplifies database interactions by providing a fluent and intuitive API. It eliminates the need for writing raw SQL queries directly and allows you to build complex database operations in a simple and elegant manner.

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();
Return $posts;
```

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

When used in conjunction with the **select()** method, the **distinct()** method applies the uniqueness constraint to the columns specified in the **select()** method.

Example:

```
$uniqueCategories = DB::table('products')
                        ->select('category')
                        ->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.

Answer:

```
$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.

Answer:

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

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

Answer:

In Laravel's query builder, the **first()** and **find()** methods are used to retrieve single records from a database table. However, there is a difference in how they are used:

1. **first()** method: The **first()** method is used to retrieve the first record that matches the given query conditions from the database table. It returns an instance of the query builder's result class or **null** if no matching record is found.
2. **find()** method: The find() method is used to retrieve a record by its primary key value. It specifically searches for a record with the given primary key in the table and returns the corresponding model or null if no record is found.

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')
                ->pluck('title');
return $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.

Answer:

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

Answer:

```
$updatedRows = DB::table('posts')
                ->where('id', 2)
                ->update([
                    'excerpt' => 'Laravel 10',
                    'description' => 'Laravel 10'
                ]);
echo "Number of affected rows: " . $updatedRows;
```

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:

```
$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.

Answer:

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

1. **`count()`:** This method calculates the total number of records that match the query criteria.

Example:

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

echo "Total number of users: " . $count;
```

In this example, the `count()` method is used to retrieve the total number of records in the "users" table.

2. **`sum()`:** This method calculates the sum of a specific column for the selected records.

Example:

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

echo "Total order amount: " . $totalAmount;
```

In this example, the `sum()` method is used to calculate the total order amount by summing the values in the "amount" column of the "orders" table.

3. **`avg()`:** This method calculates the average value of a specific column for the selected records.

Example:

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

echo "Average product price: " . $averagePrice;
```

In this example, the `avg()` method is used to calculate the average price of products by computing the mean value of the "price" column in the "products" table.

4. **`max()`:** This method retrieves the maximum value from a specific column for the selected records.

Example:

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

echo "Maximum product price: " . $maxPrice;
```

In this example, the `max()` method is used to find the maximum price of products by retrieving the highest value from the "price" column in the "products" table.

5. **`min()`:** This method retrieves the minimum value from a specific column for the selected records.

Example:

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

```
echo "Minimum product price: " . $minPrice;
```

In this example, the `min()` method is used to find the minimum price of products by retrieving the lowest value from the "price" column in the "products" table.

These aggregate methods are helpful for performing calculations and retrieving summarized information from the database without having to retrieve all the individual records.

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 allows you to add a "where not" condition to your query. It is used to retrieve records that do not match the specified condition. The method takes two arguments: the column name and the value to compare against.

Here's an example of how to use the whereNot() method:

```
$users = DB::table('users')
    ->whereNot('age', '>', 18)
    ->get();
```

In this example, we are retrieving users whose age is not greater than 18. The **whereNot()** method is used with the > operator to compare the "age" column against 18.

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:

The exists() and doesntExist() methods in Laravel's query builder are used to check the existence of records in a database table. Here's an explanation of their differences and how they are used:

exists(): The exists() method is used to check if any records exist in the specified table that match the given query criteria. It returns a boolean value (true or false) indicating whether any records were found.

doesntExist(): The doesntExist() method is the opposite of exists(). It is used to check if no records exist in the specified table that match the given query criteria. It also returns a boolean value (true or 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.

Answer:

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

Answer:

```
$id = 3;
$affectedRows = DB::table('posts')
        ->where('id', $id)
        ->increment('min_to_read');
 echo "Number of affected rows: " . $affectedRows;
```