Module 17 (Assignment)

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

   - **Laravel's query builder** is a feature that simplifies database interactions in web applications. It provides a fluent interface, allowing you to chain methods to construct queries without writing raw SQL. The query builder supports multiple database systems, handles parameter binding for security, and offers various methods for querying, joining, sorting, and aggregating data. It seamlessly integrates with the Eloquent ORM and supports transactions.

     Overall, Laravel's query builder simplifies the process of interacting with databases by providing a clean and intuitive API. It reduces the amount of boilerplate code and makes it easier to write and maintain database queries, resulting in more efficient and elegant database interactions for your Laravel applications.

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.

```php
use Illuminate\Support\Facades\DB;

$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 column or a combination of columns in the result set. It ensures that duplicate rows are eliminated, and only distinct rows are returned.
     The **distinct()** method can be used in conjunction with the **select()** method to specify which columns should be considered when determining uniqueness. By default, if no columns are specified with **distinct(),** it will consider all columns selected in the **select()** method.

Here's an example to illustrate its usage:

```php
use Illuminate\Support\Facades\DB;

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

Additionally, you can also use the **distinct()** method with multiple columns by passing an array of column names as arguments. For example

```php
$uniqueRecords = DB::table('table_name')
    ->select('column1', 'column2')
    ->distinct('column1', 'column2')
    ->get();
```

In this case, the **distinct()** method will consider the combination of "column1" and "column2" to determine uniqueness and retrieve only distinct rows based on these columns.

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. We'll store the result in the $posts variable and then print the "description" column of the $posts variable.

```php
use Illuminate\Support\Facades\DB;

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

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.

   - Here's the code to retrieve the "description" column from the "posts" table where the "id" is 2 using Laravel's query builder. We'll store the result in the $posts variable and then print the $posts variable.

```php
use Illuminate\Support\Facades\DB;

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

print_r($posts);
```

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, the **first()** and **find()** methods are used to retrieve single records from the database, but they differ in their approach and usage.

     The **first()** method is typically used when you want to retrieve the first record that matches a given set of conditions. It retrieves a single record based on the query criteria and returns it as an object. The **first()** method is commonly used when you need to retrieve a single record without specifically targeting it by its primary key.

     Here's an example of using the **first()** method:

```php
use Illuminate\Support\Facades\DB;

$post = DB::table('posts')
    ->where('status', 'published')
    ->orderBy('created_at')
    ->first();
```

   - On the other hand, the **find()** method is used to retrieve a single record by its primary key. It expects the primary key value as an argument and retrieves the record with that specific key value. The **find()** method returns the record as an object.

     Here's an example of using the **find()** method:

```php
use Illuminate\Support\Facades\DB;

$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.

    - Here's the code to retrieve the "title" column from the "posts" table using Laravel's query builder. We'll store the result in the $posts variable and then print the $posts variable.

```php
use Illuminate\Support\Facades\DB;

$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.

    - Here's the code to insert a new record into the "posts" table using Laravel's query builder with the specified values. We'll print the result of the insert operation.

```php
use Illuminate\Support\Facades\DB;

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

echo $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
use Illuminate\Support\Facades\DB;

$affectedRows = DB::table('posts')
    ->where('id', 2)
    ->update([
        'excerpt' => 'Laravel 10',
        'description' => 'Laravel 10'
    ]);

echo $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.

    - Here's the code to **delete the record** with the "id" of 3 from the "posts" table using Laravel's query builder. We'll print the number of affected rows.

```php
use Illuminate\Support\Facades\DB;

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

echo $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.

    - **The aggregate methods** in Laravel's query builder (count(), sum(), avg(), max(), and min()) are used to perform calculations on columns of a database table and retrieve aggregated results. These methods allow you to gather statistical information or compute aggregate values from a set of records.

(1) **count():** This method is used to retrieve the number of records that match the query criteria. It counts the number of rows in a table or the number of rows that satisfy the specified conditions.

Example usage of count():

```
use Illuminate\Support\Facades\DB;

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

echo $count;
```

(2) **sum():** This method calculates the sum of values in a given column. It is typically used with numeric columns to find the total sum of a specific field across the matching records.

Example usage of sum():

```
use Illuminate\Support\Facades\DB;

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

echo $totalAmount;
```

(3) **avg():** The avg() method calculates the average value of a column. It computes the arithmetic mean of the values in a specific column across the matching records.

Example usage of avg():

```
use Illuminate\Support\Facades\DB;

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

echo $averageRating;
```

(4) **max():** This method retrieves the maximum value from a column. It returns the highest value found in a specific column across the matching records.

Example usage of max():

```php
use Illuminate\Support\Facades\DB;

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

echo $maxPrice;
```

(5) **min():** The min() method retrieves the minimum value from a column. It returns the lowest value found in a specific column across the matching records.

Example usage of min():

```php
use Illuminate\Support\Facades\DB;

$minQuantity = DB::table('inventory')->min('quantity');

echo $minQuantity;
```

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

- The **whereNot()** method in Laravel's query builder is used to add a "not equal" condition to a query. It allows you to specify a column and a value that should not match in the result set.

  The **whereNot()** method takes two arguments: the column name and the value that should not match. It adds a "not equal" condition to the query, filtering out any records where the specified column's value matches the provided value.

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

```php
use Illuminate\Support\Facades\DB;

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

In this example, we're querying the "users" table and retrieving all records where the "status" column is not equal to 'active'. The **whereNot()** method is used to add the "not equal" condition to the query, filtering out any records where the "status" column value is 'active'. The **get()** method is called to retrieve the filtered records.

The resulting $users variable will contain a collection of user records that have a "status" other than 'active'.

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, but they differ in their approach and usage.
    1. **exists():** The exists() method is used to check if any records exist in the result set of a query. It returns true if at least one record is found, and false if no records are found.

       Example usage of **exists():**

```php
use Illuminate\Support\Facades\DB;

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

if ($exists) {
    // Records exist
} else {
    // No records found
}
```

2. **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 true if no records are found, and false if at least one record is found.

   Example usage of **doesntExist():**

```php
use Illuminate\Support\Facades\DB;

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

if ($doesntExist) {
    // No records found
} else {
    // Records exist
}
```

To summarize, **exists()** is used to check if any records exist, while **doesntExist()** is used to check if no records exist. They provide a convenient way to validate the presence or absence of records in a table based on certain conditions, allowing you to perform different actions accordingly.

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
use Illuminate\Support\Facades\DB;

$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.

    - 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
use Illuminate\Support\Facades\DB;

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

echo $affectedRows;
```