Title: Assignment on Query Builder

Name: Saidul Islam

Module: 17

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 fluent interface for creating and executing database queries in Laravel. It provides a simple and elegant way to interact with databases by allowing developers to write database queries using chainable methods instead of writing raw SQL statements. The query builder abstracts the underlying database system, making it easier to switch between different database engines without changing the code and it also uses PDO parameter binding to protect our application from SQL injection attacks

Let's say we have a "'users'" table with the following columns: id, name, email, and created at.

To retrieve all users from the table, we can use the following code:

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

      foreach ($users as $user) {
          echo $user->name;
      }
```

In this example, we use the DB facade to access the query builder. The table method specifies the table we want to query (in this case, 'users'), and the get method retrieves all records from that table. We can then iterate over the results and access each 'users' properties, such as their name.

The query builder also provides methods for filtering, sorting, and limiting results. For example, to retrieve users with a specific email domain, we can add a where condition.

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

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.

Route:

```
Route::get('/excerptDescription',[PostController::class,'excerptDescription']);
```

Controller Method:

```
    public function excerptDescription(){

        $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?

The distinct() method in Laravel's query builder is used to retrieve unique records from a table. It ensures that duplicate rows are eliminated from the result set.

distinct() is used in conjunction with the select() method to specify the columns to be retrieved and ensure uniqueness across those columns.

Example:

```
    public function distinctSelect()
    {
        $posts = DB::table('posts')->select('min_to_read')->distinct()->get();
        return $posts;
    }
```

This will return only distinct rows as a result.

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.

Route:

```
Route::get('/retriveFirst',[PostController::class,'retriveFirst']);
```

Controller Method:

```
    public function retriveFirst()
    {
        $posts = DB::table('posts')
            ->where('id', 2)
            ->first();
        return $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.

Route:

```
Route::get('/description',[PostController::class,'description']);
```

Controller Method:

```
    public function description()
    {
        $posts = DB::table('posts')
            ->where('id', 2)
            ->value('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?

The first() method in Laravel's query builder retrieves the first record that matches the query conditions and returns it as an object. It is typically used when you expect only one record to be returned.

Example:

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

The find() method, on the other hand, retrieves a record by its primary key. It is used when you know the primary key value of the record you want to retrieve.

Example:

```
  $posts = DB::table('posts')->find(2);
        return $posts;
```

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.

Route:

```
Route::get('/title',[PostController::class,'title']);
```

Controller Method:

```
    public function title()
    {
        $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.

Route:

```
Route::post('/insertData',[PostController::class,'insertData']);
```

Controller Method:

```
public function insertData()
    {
        $posts = DB::table('posts')->insert([
            'title' => 'X',
            'slug' => 'X',
            'excerpt' => 'excerpt',
            'description' => 'description',
            'is_published' => true,
            'min_to_read' => 2,
        ]);
        return $posts;
    }
```

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.

Route:

```
Route::patch('/update ',[PostController::class,'update']);
```

Controller Method:

```
    public function update()
    {
        $update = DB::table('posts')
            ->where('id',  2)
            ->update([
                'excerpt'     => 'Laravel 10',
                'description' => 'Laravel 10',
            ]);
        return $update;
    }
```

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.

Route:

```
Route::delete ('/delete  ',[PostController::class,'delete']);
```

Controller Method:

```
    public function delete()
    {
        $delete = DB::table('posts')
            ->where('id', 3)
            ->delete();
        return $delete;
    }
```

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 a set of records returned from a database query. These methods allow us to retrieve aggregated values from columns, such as counting the number of records, calculating the sum or average of a numeric column, or finding the maximum and minimum values within a column.

1. count(): It returns the number of records in a table or the number of records that match a specific condition

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

In this example, the count() method is used to get the total number of records in the 'posts'table.

2. sum(): This method calculates the sum of a specific column's values. It is typically used
on numeric columns..

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

This example calculates the total sales by summing the values in the "amount" column of the 'orders' table.

3. avg(): It calculates the average of values in a numeric column.

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

Here, the avg() method calculates the average price from the 'products' table

4. max(): Returns the maximum value of a specific column in a table.

```
  $max = DB::table('employees')->max('salary');
        echo $max;
```

Here, the max() method calculates the maximum salary from 'employees' table

5. min(): Returns the minimum value of a specific column in a table.

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

Here, the min() method calculates the minimum salary from 'products'table.

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" condition to a query. It allows you to retrieve records that do not match a specific condition. Here's an example of its usage:

```
       $posts = DB::table('posts')
            ->whereNot('status', 'published')
            ->get();
        return $posts;
```

In this example, the query will retrieve all records from the 'posts' table where the 'status' column is not equal to 'published'.

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?

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

exists(): The exists() method returns true if the query has at least one matching record, and false otherwise. It is used to verify if any records exist that satisfy the specified conditions.
Example:

```
   $exists = DB::table('posts')
    ->where('status', 'published')
    ->exists();
if ($exists) {
    echo "published post exists.";
} else {
    echo "No published posts found.";
}
```

doesntExist(): The doesntExist() method returns true if the query does not have any matching records, and false otherwise. It is used to check if no records exist that satisfy the specified conditions.

Example:

```
   $doesntExist = DB::table('posts')
    ->where('status', 'published')
    ->doesntExist();
if ($doesntExist) {
    echo "No published posts found.";
} else {
    echo "At least one published post exists.";
}
```

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.

Route:

```
Route::get ('/minData',[PostController::class,'minData']);
```

Controller Method:

```
    public function minData()
    {
        $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.

Route:

```
Route::get ('/incrementByOne  ',[PostController::class,'incrementByOne']);
```

Controller Method:

```
    public function incrementByOne()
    {
        $posts = DB::table('posts')
            ->where('id', 3)
            ->increment('min_to_read');
        return $posts;
    }
```