MODULE - 17

Submitted by : **Sheikh Mohammed Morshed**

Date : 12/06/2023

ASSIGNMENT :

Instructions:

- Read the given code snippets and questions carefully.
- Write the code that best answers each question.
- Make sure to provide clear and concise answers.
- Submit your completed assignment within the given time frame.

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 of the Laravel framework that provides a convenient and intuitive way to interact with databases. It allows developers to build database queries using a fluent, chainable interface, making it easier to create and modify database queries compared to writing raw SQL.

   The query builder in Laravel is designed to work with multiple database management systems (DBMS) such as MySQL, PostgreSQL, SQLite, and SQL Server, among others. It abstracts the underlying database system, allowing developers to write database-agnostic code.

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

   1. Fluent Interface: The query builder utilizes a fluent interface, which means you can chain methods together to construct complex queries in a readable and expressive manner. This makes the code more readable and reduces the need for manual concatenation of SQL strings.

   2. Concise Syntax: Laravel's query builder provides a concise syntax that resembles writing SQL queries, but with the added benefits of object-oriented programming. It offers methods for common database operations such as selecting, inserting, updating, and deleting records, as well as for joining tables, grouping, sorting, and aggregating data.

   3. Parameter Binding: The query builder handles parameter binding automatically, helping to prevent SQL injection attacks. You can pass parameters directly into the query builder methods, and Laravel will handle the proper escaping and binding of the values.

   4. Query Building Methods: Laravel's query builder provides a wide range of methods for building queries. For example, you can use methods like `select()`, `where()`, `orderBy()`, `join()`, `groupBy()`, and `having()` to construct sophisticated queries without the need to write complex SQL statements manually.

5. Eloquent Integration: Laravel's query builder seamlessly integrates with Eloquent, Laravel's Object-Relational Mapping (ORM) system. This means you can use the query builder to build queries for your Eloquent models, allowing you to take advantage of both the query builder's flexibility and Eloquent's powerful features, such as model relationships and attribute casting.

Overall, Laravel's query builder simplifies database interactions by providing an expressive and intuitive API. It reduces the need for manual SQL string manipulation, enhances code readability, and promotes best practices like parameter binding and security. These features make it a popular choice for developers working with Laravel to interact with databases in a simple and elegant way.

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 an example code snippet that retrieves the "excerpt" and "description" columns from the "posts" table using Laravel's query builder:

```
*/
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 ensure that only unique values are returned from a query. It eliminates duplicate rows from the result set, so each row in the result will have distinct values for the selected columns.

The distinct() method is typically used in conjunction with the select() method to specify the columns for which uniqueness should be enforced. By default, the select() method retrieves all columns from the table, but when used with distinct(), it allows you to narrow down the selection to specific 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 an example code snippet that retrieves the first record from the "posts" table where the "id" is 2 using Laravel's query builder:

```
*/
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 an example code snippet that retrieves 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 :

```
*/
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?**

   The first() and find() methods in Laravel's query builder are both used to retrieve a single record from a table, but they have some differences in their usage and behavior:

   first(): The first() method retrieves the first record that matches the query conditions. It returns a single object representing the first matching record. It is commonly used when you want to retrieve the first record based on a specific condition or when you need to retrieve a single record without specifying an exact identifier.

   find(): The find() method is used to retrieve a record by its primary key value. It takes the primary key value as an argument and returns the corresponding record as an object. It is particularly useful when you know the exact identifier (primary key value) of the record you want to retrieve.

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 an example code snippet that retrieves the "title" column from the "posts" table using Laravel's query builder:

```
*/
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 an example code snippet that inserts a new record into the "posts" table using Laravel's query builder:

```
*/
use Illuminate\Support\Facades\DB;
$data = [
    'title' => 'X',
    'slug' => 'X',
    'excerpt' => 'excerpt',
    'description' => 'description',
    'is_published' => true,
    'min_to_read' => 2,
];
$result = DB::table('posts')->insert($data);
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 an example code snippet that inserts a new record into the "posts" table using Laravel's query builder:

```
*/
use Illuminate\Support\Facades\DB;
$data = [
    'title' => 'X',
    'slug' => 'X',
    'excerpt' => 'excerpt',
    'description' => 'description',
    'is_published' => true,
    'min_to_read' => 2,
];
$result = DB::table('posts')->insert($data);
echo $result;
/*
```

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 an example code snippet that deletes the record with the "id" of 3 from the "posts" table using Laravel's query builder:

```
*/
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.

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

```
*/
use Illuminate\Support\Facades\DB;
$totalUsers = DB::table('users')->count();
/*
```

sum(): The sum() method calculates the sum of a specified column's values. It returns the sum as a numeric value :

```
*/
use Illuminate\Support\Facades\DB;
$totalAmount = DB::table('orders')->sum('amount');
/*
```

avg(): The avg() method computes the average value of a specified column's values. It returns the average as a numeric value :

```
*/use Illuminate\Support\Facades\DB;
$averageRating = DB::table('reviews')->avg('rating');
/*
```

max(): The max() method retrieves the maximum value from a specified column. It returns the maximum value as the same data type as the column :

```
*/
use Illuminate\Support\Facades\DB;
$maxPrice = DB::table('products')->max('price');
/*
```

min(): The min() method retrieves the minimum value from a specified column. It returns the minimum value as the same data type as the column :

```
*/
use Illuminate\Support\Facades\DB;
$minStock = DB::table('products')->min('stock');
/*
```

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 "where not" condition to a query. It allows you to retrieve records that do not match a specific condition.

    Example of whereNot():

```
*/
use Illuminate\Support\Facades\DB;
$users = DB::table('users')
   ->where('status', '=', 'active')
   ->whereNot('role', 'admin')
   ->get();
/*
```

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 or non-existence of records in a table based on a specific condition. Here's an explanation of their differences and usage:

    exists(): The exists() method is used to check if any records exist in the table that match a specific condition. It returns a boolean value indicating whether the query returns any records or not.

```
*/
use Illuminate\Support\Facades\DB;
$exists = DB::table('users')->where('status', 'active')->exists();
/*
```

    doesntExist(): The doesntExist() method is used to check if no records exist in the table that match a specific condition. It returns a boolean value indicating whether the query returns zero records or not.

```
*/
use Illuminate\Support\Facades\DB;
$doesntExist = DB::table('users')->where('status', 'deleted')->doesntExist();
/*
```

    Both exists() and doesntExist() methods are useful for performing conditional checks based on the existence of records. They can be used in various scenarios such as validating unique entries, checking for related records, or verifying if a specific condition is met in the database.

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 an example code snippet that retrieves records from the "posts" table where the "min_to_read" column is between 1 and 5 using Laravel's query builder:

```
*/
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 an example code snippet that increments 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:

```
*/
use Illuminate\Support\Facades\DB;
$affectedRows = DB::table('posts')
    ->where('id', 3)
    ->increment('min_to_read', 1);
echo $affectedRows;
/*
```