Ostad Module-7 Assignment

Q1: 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 of the Laravel framework that provides a convenient and expressive way to interact with databases. It allows you to build and execute database queries using a fluent and chainable interface, making database operations more readable and maintainable.

The query builder abstracts the underlying database system, providing a consistent API for working with different database engines, such as MySQL, PostgreSQL, SQLite, and others. This means you can write database queries in a vendor-agnostic manner, and Laravel will handle the necessary translations to the appropriate SQL syntax for the target database.

Here are some key features and benefits of Laravel's query builder:

1. Fluent interface: The query builder uses a fluent interface, which means you can chain multiple method calls together to construct a query. This leads to more readable and concise code compared to writing raw SQL strings.

2. Parameter binding: Laravel's query builder automatically handles parameter binding, preventing SQL injection vulnerabilities. You can pass user input as placeholders in your queries, and Laravel will sanitize and escape the values appropriately.

3. Eloquent integration: Laravel's ORM, called Eloquent, is built on top of the query builder. This integration allows you to seamlessly switch between using the query builder and the more advanced Eloquent ORM, depending on your needs.

4. Query building methods: The query builder provides a wide range of methods for constructing different types of queries. You can easily perform common operations such as selecting columns, filtering data using conditions (where clauses), joining tables, ordering results, grouping, and aggregating data.

5. Database agnostic: With the query builder, you can write code that is not tightly coupled to a specific database system. This makes it easier to switch between different database engines or support multiple databases in the same application.

6. Migration support: Laravel's query builder integrates with Laravel's migration system. You can define database table structures and modifications using migration files, and the query builder helps you execute those migrations to keep your database schema in sync with your application code.

Overall, Laravel's query builder simplifies database interactions by providing a clean and expressive API. It reduces the need to write raw SQL queries, promotes code readability, and ensures secure database operations.

Q2: 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:

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

    $posts = DB::table('posts')
        ->select('excerpt', 'description')
        ->get();

    print_r($posts);
==========================================
```

N.B.: we first import the DB facade, which provides access to the query builder. Then, we use the table() method to specify the "posts" table as the target of our query.

Q3: 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 column or set of columns in a database query. It ensures that the result set contains only distinct (unique) values, eliminating any duplicates.

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

    $uniqueTitles = DB::table('posts')
                ->select('title')
                ->distinct()
                ->get();

    print_r($uniqueTitles);
==========================================
```

Above code snippet, we are querying the "posts" table and retrieving distinct values from the "title" column. The select('title') method specifies that we only want to retrieve the "title" column from the table. Then, we chain the distinct() method to ensure that only unique titles are included in the result set.

The distinct() method is particularly useful when you want to eliminate duplicate values from the result set and work with only unique values. It can be used with multiple columns as well by passing an array of column names to the select() method.

Q4: 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:

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

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

    print_r($posts->description);
==========================================
```

Q5: 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:

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

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

    print_r($posts);
==========================================
```

Q6: Explain the difference between the first() and find() methods in Laravel's query builder. How are they used to retrieve single records?

Answer: The first() method retrieves the first record based on specified conditions or the default order, while the find() method retrieves a record by its primary key value. The first() method is more flexible in terms of conditions and can be used when the primary key is unknown, whereas the find() method is specifically used to fetch records by their primary key value.

Example the first() Method –

```
==========================================================
    $post = DB::table('posts')->where('category', 'News')->first();
==========================================================
```

Example the find() Method –

```
==========================================================
    $post = DB::table('posts')->find(1);
==========================================================
```

Q7: 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:

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

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

    print_r($posts);
==========================================
```

Q8: 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:

```
==========================================
    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
        ]);

    print_r($result);
==========================================
```

Q9: 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:

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

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

    echo "Number of affected rows: " . $affectedRows;
=======================================================
```

Q10: 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:

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

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

    echo "Number of affected rows: " . $affectedRows;
=======================================================
```

Q11: 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:

count() – The count() method is used to retrieve the number of records that match the specified conditions. Example –

```
$count = DB::table('users')->where('active', true)->count();
```

sum() – The sum() method is used to calculate the sum of a specific column's values. Example –

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

avg() – The avg() method is used to calculate the average (mean) value of a specific column's values. Example –

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

max() – The max() method is used to retrieve the maximum value from a specific column. Example –

```
$highest = DB::table('scores')->max('score');
```

min() – The min() method is used to retrieve the minimum value from a specific column. Example –

```
$lowest = DB::table('prices')->min('amount');
```

Q12: Describe how the whereNot() method is used in Laravel's query builder. Provide an example of its usage.

Answer: The whereNot() method can be used in two different ways:

a) Comparison with a single value:

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

b) Comparison with an array of values:

```
$query = DB::table('products')
        ->whereNotIn('category', ['electronics', 'clothing'])
        ->get();
```

Q13: 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:

exists() method: The exists() method is used to check if any records exist in the specified table that match the given conditions. It returns ***true*** if at least one record is found, and ***false*** otherwise. It is typically used when we want to verify if there are any records that meet certain criteria before performing further actions. Exapmple –

```
==============================================================
    $exists = DB::table('products')->where('category', 'electronics')->exists();
==============================================================
```

doesntExists() method: The doesntExists() method is used to check if no records exist in the specified table that match the given conditions. It returns ***true*** if no records are found, and ***false*** if there is at least one matching record. It is commonly used when we want to ensure that no records exist before proceeding with certain operations. Exapmple –

```
==========================================================================
    $doesntExist = DB::table('products')->where('category', 'books')->doesntExist();
==========================================================================
```

Q14: 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:

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

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

    print_r($posts);
==========================================
```

Q15: 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:

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

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

    echo "Number of affected rows: " . $affectedRows;
================================================
```

Github repository link: https://github.com/mahmudcdh/OstadAssignment_Module17.git