Assinment Ostad PHP-LARAVEL Batch

Md. Asadullah Hil Galib

Answer to the question no 01:

Laravel's query builder is a feature of the Laravel framework that provides a convenient and expressive way to interact with databases using a fluent, object-oriented interface. It allows developers to build and execute database queries using method chaining and expressive syntax, making database interactions in Laravel applications more intuitive and readable.

Here are some key aspects of Laravel's query builder and how it simplifies database interactions:

Fluent Interface: The query builder uses a fluent interface, which means that each method call returns a new query builder instance. This allows for method chaining, enabling developers to build complex queries in a readable and concise manner.

Database Agnostic: Laravel's query builder is database agnostic, meaning it supports multiple database systems. It provides a consistent API for interacting with databases, regardless of the underlying database engine. This makes it easier to switch between different databases without changing the query syntax.

Parameter Binding: The query builder supports parameter binding, which helps prevent SQL injection attacks and improves security. Parameters are automatically bound to the query, eliminating the need for manual sanitization of user input.

Readability and Maintainability: The query builder's expressive syntax makes database queries more readable and easier to understand. It uses method names that closely resemble SQL keywords, making the code more self-explanatory and reducing the need to write raw SQL queries.

Query Building and Execution: The query builder simplifies the process of constructing queries by providing methods for various SQL clauses, such as select(), where(), orderBy(), join(), and more. It abstracts away the complexities of SQL syntax and generates the appropriate SQL statements based on the methods called. Once the query is built, it can be executed using the get(), first(), or other result retrieval methods.

Integration with Eloquent ORM: The query builder seamlessly integrates with Laravel's Eloquent ORM, allowing for easy integration between raw queries and Eloquent models. This combination provides a powerful toolkit for database operations, giving developers flexibility in choosing between raw queries or utilizing the ORM.

Overall, Laravel's query builder provides a simpler and more elegant way to interact with databases in Laravel applications. It offers a robust set of methods, an expressive syntax, and a seamless integration with the framework's ecosystem, contributing to efficient and maintainable database interactions.

Answer to the question no 02:

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

Answer to the question no 03:

The distinct() method in Laravel's query builder is used to retrieve only unique values from a column in the database table. It ensures that the result set contains distinct (non-duplicate) values.

When used in conjunction with the select() method, the distinct() method modifies the query to apply the uniqueness constraint on the specified column(s). It narrows down the result set by removing duplicate values from the selected column(s).

Answer to the question no 04:

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

```
if ($posts) {
   echo $posts->description;
} else {
   echo "No record found.";
}
```

Answer to the question no 05:

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

print_r($posts);
```

Answer to the question no 06:

The first() and find() methods in Laravel's query builder are both used to retrieve a single record from the database. However, there are some differences in how they are used and the conditions they handle:

first() method: The first() method is used to retrieve the first record that matches the query conditions. It does not require specifying the primary key explicitly. When using first(), you typically chain it after the query builder methods that define the conditions, such as where() or orderBy().

example:

```
$firstPost = DB::table('posts')
        ->where('category', 'News')
        ->orderBy('created_at', 'desc')
        ->first();
```

In this example, first() is used to retrieve the first record from the "posts" table where the "category" column is 'News', ordered by the "created_at" column in descending order.

find() method: The find() method is used to retrieve a record by its primary key value. You need to explicitly specify the primary key value as an argument to the find() method.

example:

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

In this example, find(1) is used to retrieve the record from the "posts" table with the primary key value of 1.

Answer to the question no 07:

```
$posts = DB::table('posts')
        ->pluck('title');


print_r($posts);
```

Answer to the question no 08:

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


if ($result) {
```

```
    echo "New record inserted successfully!";
} else {
    echo "Failed to insert new record.";
}
```

Answer to the question no 09:

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

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

Answer to the question no 10:

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

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

Answer to the question no 11:

The aggregate methods (count(), sum(), avg(), max(), and min()) in Laravel's query builder allow you to perform calculations on a column or set of columns in a database table. Here's a brief explanation and an example of each:

count(): The count() method is used to retrieve the number of rows in a table or the number of rows that match specific conditions. It returns a single value representing the count. Here's an example:

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

echo "Total number of posts: " . $totalPosts;
```

This example retrieves the total number of posts in the "posts" table.

sum(): The sum() method is used to calculate the sum of a column's values. It returns the total sum as a single value. Here's an example:

```
$totalViews = DB::table('posts')->sum('views');

echo "Total views of all posts: " . $totalViews;
```

This example calculates the total sum of the "views" column in the "posts" table.

avg(): The avg() method is used to calculate the average value of a column. It returns the average as a single value. Here's an example:

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

echo "Average price of products: " . $averagePrice;
```

This example calculates the average price of the products in the "products" table.

max(): The max() method is used to find the maximum value in a column. It returns the maximum value as a single value. Here's an example:

```
$maxAge = DB::table('users')->max('age');

echo "Maximum age among users: " . $maxAge;
```

This example finds the maximum age among the users in the "users" table.

min(): The min() method is used to find the minimum value in a column. It returns the minimum value as a single value. Here's an example:

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

echo "Minimum price of products: " . $minPrice;
```

This example finds the minimum price among the products in the "products" table.

These aggregate methods provide a convenient way to perform calculations on columns in a table and retrieve summary information from the database.

Answer to the question no 12:

The whereNot() method in Laravel's query builder is used to add a "not equal" condition to the query. It allows you to retrieve records where a specific column's value is not equal to a given value.

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

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

In this example, we are retrieving records from the "users" table where the "status" column is not equal to 'active'. The whereNot() method takes two arguments: the column name and the value to compare against.

Answer to the question no 13:

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

exists(): The exists() method is used to check if at least one record exists in the query result. It returns a boolean value indicating whether any records match the specified conditions. Here's an example:

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

if ($exists) {
```

```
    echo "At least one active user exists.";
} else {
    echo "No active users found.";
}
```

In this example, the exists() method is used to check if there is at least one active user in the "users" table. If the query result has one or more matching records, the exists() method will return true, indicating that at least one active user exists. Otherwise, if there are no matching records, it will return false.

doesntExist(): The doesntExist() method is the negation of exists(). It is used to check if no records exist in the query result. It returns a boolean value indicating whether there are no records that match the specified conditions. Here's an example:

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

if ($doesntExist) {
    echo "No active users found.";
} else {
    echo "At least one active user exists.";
}
```

In this example, the doesntExist() method is used to check if there are no active users in the "users" table. If the query result has no matching records, the doesntExist() method will return true, indicating that no active users are found. Otherwise, if there is at least one matching record, it will return false.

Answer to the question no 14:

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

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

Answer to the question no 15:

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

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