Assignment: Query Builder in Laravel

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

Ans: Fluent and expressive syntax: The query builder uses a fluent interface, allowing you to chain methods together to construct queries. This results in a readable and expressive syntax that closely resembles the structure of SQL statements. The fluent syntax makes it easier to understand and write queries, especially for complex operations involving multiple conditions and joins.

Database agnostic: The query builder is designed to work with multiple database systems, such as MySQL, PostgreSQL, SQLite, and SQL Server. It abstracts the underlying differences between these databases, allowing you to write database-agnostic code. You can switch between database systems without having to change your code, which improves flexibility and reduces vendor lock-in.

Parameter binding: Laravel's query builder automatically handles parameter binding. When you pass values into queries, the query builder properly escapes and sanitizes them, preventing SQL injection attacks. Parameter binding improves security by separating the data from the query structure.

Query building methods: Laravel's query builder provides a comprehensive set of methods for constructing queries. You can use methods like select, where, join, orderBy, groupBy, limit, and offset to build queries with various conditions, joins, sorting, grouping, and pagination. These methods encapsulate the necessary SQL syntax, making it easier to construct queries without dealing with the low-level details.

Eloquent ORM integration: The query builder seamlessly integrates with Laravel's Eloquent ORM, a powerful object-relational mapping tool. Eloquent allows you to define models that represent database tables, and you can use the query builder to construct queries on these models. This integration provides additional features like model relationships, eager loading, and attribute casting, making it even more convenient to interact with databases.

Query logging and debugging: Laravel's query builder offers built-in query logging, which means you can log and review the SQL queries executed by your application. This feature is particularly useful for debugging and optimizing database interactions. You can easily identify slow or problematic queries and make necessary optimizations.

Subqueries: The query builder supports subqueries, allowing you to nest queries within other queries. This feature is handy when you need to perform complex operations or retrieve data based on advanced conditions. Subqueries help you build more sophisticated queries while maintaining the readability and maintainability of your code.

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.

Ans: **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?

Ans: **use Illuminate\Support\Facades\DB;**

**$uniqueNames = DB::table('users')**

**->select('name')**

**->distinct()**

**->get();**

**print\_r($uniqueNames);**

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.

Ans:

**use Illuminate\Support\Facades\DB;**

**$posts = DB::table('posts')**

**->where('id', 2)**

**->first();**

**if ($posts) { echo $posts->description;**

**} else { echo "No record found.";**

**}**

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.

Ans:

**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?

**use Illuminate\Support\Facades\DB;**

**$firstPost = DB::table('posts')**

**->where('published', true)**

**->orderBy('created\_at')**

**->first();**

Then:

**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.

**Ans: 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.

Ans:

**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**

**]);**

**if ($result) { echo "Record inserted successfully.";**

**} else { echo "Failed to insert record.";**

**}**

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.

Ans:

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;

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.

Ans: use Illuminate\Support\Facades\DB;

$affectedRows = DB::table('posts')

->where('id', 3)

->delete();

echo "Number of affected rows: " . $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.

use Illuminate\Support\Facades\DB;

$totalCount = DB::table('posts')->count(); echo "Total number of posts: " . $totalCount;

use Illuminate\Support\Facades\DB;

$totalAmount = DB::table('orders')->sum('amount'); echo "Total amount of orders: " . $totalAmount;

use Illuminate\Support\Facades\DB;

$averageRating = DB::table('reviews')->avg('rating'); echo "Average rating: " . $averageRating;

use Illuminate\Support\Facades\DB;

$highestPrice = DB::table('products')->max('price'); echo "Highest price: " . $highestPrice;

use Illuminate\Support\Facades\DB;

$lowestPrice = DB::table('products')->min('price'); echo "Lowest price: " . $lowestPrice;

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

use Illuminate\Support\Facades\DB;

$users = DB::table('users')

->whereNot('status', 'active')

->get();

SELECT \* FROM users WHERE status <> '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?

use Illuminate\Support\Facades\DB;

$exists = DB::table('users')

->where('status', 'active')

->exists();

if ($exists) { echo "There are active users.";

} else { echo "No active users found.";

}

use Illuminate\Support\Facades\DB;

$doesntExist = DB::table('users')

->where('status', 'active')

->doesntExist();

if ($doesntExist) { echo "No active users found.";

} else {

echo "There are active users.";

}

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.

Ans:

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.

use Illuminate\Support\Facades\DB;

$affectedRows = DB::table('posts')

->where('id', 3)

->increment('min\_to\_read'); echo "Number of affected rows: " . $affectedRows;