Assignment 17

1. Laravel's database query builder provides a convenient, fluent interface to creating and running database queries. It can be used to perform most database operations in your application and works perfectly with all of Laravel's supported database systems.
   The Laravel query builder uses PDO parameter binding to protect your application against SQL injection attacks. There is no need to clean or sanitize strings passed to the query builder as query bindings.

2. Certainly! Here's an example of how we can use Laravel's query builder to retrieve the "excerpt" and "description" columns from the "posts" table and store the result in the **$posts** variable:

   ```
   use Illuminate\Support\Facades\DB; // ...

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

   ->select('excerpt', 'description')

   ->get();

   print_r($posts);
   ```

   In the code above, we import the **DB** facade from Laravel's **Illuminate\Support\Facades** namespace. Then, we use the **table()** method on the **DB** facade to specify the "posts" table. Next, we chain the **select()** method to specify the columns we want to retrieve, in this case, "excerpt" and "description".

   Finally, we use the **get()** method to execute the query and retrieve the results. The resulting records will be stored in the **$posts** variable, which we then print using **print_r()**.

   Make sure you have the necessary Laravel dependencies installed and configured, and that you have a connection to your database set up in your Laravel application before executing this code.

3. The distinct() method in Laravel's query builder is used to retrieve only unique records from a table or query result. It allows you to eliminate duplicate rows and retrieve distinct values based on the specified columns.

   When used in conjunction with the select() method, the distinct() method ensures that the selected columns are evaluated for uniqueness. By default, the select() method

retrieves all columns specified in the query, including any duplicates. However, if you want to fetch only the unique values of specific columns, you can use the distinct() method.

Here's an example to illustrate the usage of distinct() in conjunction with the select() method:

```
use Illuminate\Support\Facades\DB;// ...
$uniqueTitles = DB::table('posts')
   ->select('title')
   ->distinct()
   ->get();
print_r($uniqueTitles);
```

In the code above, we use the **distinct()** method after the **select()** method to retrieve only unique values for the "title" column from the "posts" table. The **get()** method executes the query and retrieves the distinct titles.
By using **distinct()** in this example, if there are multiple rows with the same title in the "posts" table, only one instance of each unique title will be returned in the result set.
The **distinct()** method is particularly useful when you need to eliminate duplicate values and retrieve only the unique records from a specific column or combination of columns in your database query.

4.
```
use Illuminate\Support\Facades\DB;// ...
$posts = DB::table('posts')
->where('id', 2)
->first();
  if ($posts) {
     echo $posts->description;
  } else {
     echo "No post found with ID 2.";
  }
```

5.
```
use Illuminate\Support\Facades\DB;// ...
```

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

6. **first() method:**

The first() method is used to retrieve the first record that matches the specified conditions.

It is typically used in conjunction with methods like where() to filter the records based on specific criteria.

If a record is found, the first() method returns a single object representing that record.

If no record is found, it returns null.

Example:

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

find() method:

The find() method is used to retrieve a record by its primary key value.

It assumes that the primary key column in the table is named id.

The find() method takes the primary key value as its argument and returns the corresponding record.

If a record is found, the find() method returns a single object representing that record.

If no record is found, it returns null.

Example:

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

7.

```
use Illuminate\Support\Facades\DB;// ...
$posts = DB::table('posts')
 ->pluck('title');
print_r($posts);
```

8.

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

9.

```
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.

```
use Illuminate\Support\Facades\DB;// ...
$affectedRows = DB::table('posts')
   ->where('id', 3)
   ->delete();
 echo "Number of affected rows: " . $affectedRows;
```

11.

count():

The count() method is used to retrieve the total count of rows that match the specified conditions.

Example:

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

sum():

The sum() method is used to retrieve the sum of values in a specific column.

Example:

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

avg():

The avg() method is used to retrieve the average value of a specific column.

Example:

```
$averageRating = DB::table('reviews')->avg('rating');
```

max():

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

Example:

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

min():

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

Example:

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

12.

In Laravel's query builder, the whereNot() method is used to add a "not equal" condition to a query. It allows you to specify a column and a value that should not match in the query results.

Here's an explanation of how the whereNot() method is used and an example to illustrate its usage:

The whereNot() method accepts two parameters:

The column name: The name of the column on which the "not equal" condition should be applied.

The value: The value that should not match the column in the query results.

Example: Suppose we have a "users" table with columns "id" and "status". We want to retrieve all users whose status is not equal to 'active'. We can use the whereNot() method to achieve this:

```
use Illuminate\Support\Facades\DB;// ...

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

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

  ->get();
```

13.

In Laravel's query builder, the exists() and doesntExist() methods are used to check the existence of records in a table. They provide a convenient way to determine whether a query result contains any records or not.

Here's an explanation of the difference between the two methods and how they are used:

exists() method:

The exists() method is used to check if any records exist in the query result.

It returns true if at least one record exists, and false if no records are found.

Example:

```
$hasRecords = DB::table('users')->where('role', 'admin')->exists();
```

doesntExist() method:

The doesntExist() method is used to check if no records exist in the query result.

It returns true if no records are found, and false if at least one record exists.

Example:

```
$hasNoRecords = DB::table('users')->where('role', 'admin')->doesntExist();
```

14.

```
use Illuminate\Support\Facades\DB;// ...
$posts = DB::table('posts')
   ->whereBetween('min_to_read', [1, 5])
   ->get();
 print_r($posts);
```

15.

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

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