1. 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 powerful feature that provides a convenient and expressive way to interact with databases in your application. It allows you to construct database queries using a fluent, chainable interface, making it easier to fetch and manipulate data. Laravel's query builder offers a higher-level abstraction over raw SQL, enabling you to write database queries in a more readable and maintainable manner. Laravel's query builder provides a wide range of methods to construct complex queries effortlessly. Laravel's query builder supports parameter binding, which helps protect your application against SQL injection attacks. In addition to retrieving data, Laravel's query builder also perform data manipulation operations like inserting, updating, and deleting records using similar fluent methods. Laravel provides a wide range of methods and well-documented. Overall, Laravel's query builder simplifies database interactions by providing a clean and elegant syntax, protecting against common security risks, and offering a higher level of abstraction over raw SQL queries. It greatly enhances developer productivity and code readability, making it one of the standout features of the Laravel framework.

```php
$users = DB::table('users')
            ->where('age', '>', 25)
            ->get();
```

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.

Answer:-

```php
class PostsController extends Controller
{
    public function getPosts() {
        $posts = DB::table('posts')
            ->select('excerpt', 'description')
            ->get();

        return $posts;
    }
}
```

```php
Route::get('/posts',[ PostsController::class, 'getPosts']);
```

3. 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 only distinct (unique) values from a specific column or a combination of columns in the database table. It helps eliminate duplicate values and ensures that each returned row is unique based on the selected column(s).

When used in conjunction with the `select()` method, the `distinct()` method applies the distinct operation to the columns specified in the `select()` method.

```php
public function postDistinct() {
    $uniqueTitle = DB::table('posts')
            ->select('title')
            ->distinct()
            ->get();

    return $uniqueTitle;
}
```

```php
Route::get('/postdistinct',[ PostsController::class, 'postDistinct']);
```

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.

Answer:-

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

```
Route::get('/firstrecord',[ PostsController::class, 'firstRecord']);
```

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.

```
//Answer of question no 5
public function postDescription() {
    $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?

Answer:- In Laravel's query builder, both the `first()` and `find()` methods are used to retrieve single records from a database table. However, there are key differences in their usage and behavior:

`first()`: The `first()` method retrieves the first record that matches the given conditions or the first record in the table if no conditions are specified. It returns a single object representing the retrieved row. Here's an example:

`find()`: The `find()` method retrieves a record by its primary key. It expects the primary key value as its argument and returns a single object representing the matching record.

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.

```
    //Answer of question no 6
    public function postTitle() {
        $posts = DB::table('posts')
            ->pluck('title');

        return $posts;
    }
```

```
Route::get('/posttitle',[ PostsController::class, 'postTitle']);
```

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.

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

    print_r($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.

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

    return $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.

```php
//Answer of question no 10
public function deleteRows() {
    $deleteRows = DB::table('posts')
            ->where('id', 3)
            ->delete();

    print_r($deleteRows);

}
```

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.

Answer:- In Laravel's query builder, the aggregate methods `count()`, `sum()`, `avg()`, `max()`, and `min()` are used to perform calculations on a set of records in a database table. These methods provide a convenient way to retrieve aggregated values based on specific columns or conditions.

```php
//Answer of question no 11
public function aggreGatesCount(){
    $result =DB::table('products')->count();
    return $result;
}
public function aggreGatesMax(){
    $result =DB::table('products')->max('price');
    return $result;

}
public function aggreGatesMin(){
    $result =DB::table('products')->min('price');
    return $result;

}
public function aggreGatesAvg(){
    $result =DB::table('products')->avg('price');
    return $result;

}
public function aggreGatesSum(){
    $result =DB::table('products')->sum('price');
    return $result;

}
```

```php
//!Aggregates
Route::get('/count', [PostsController::class, 'aggreGatesCount']);
Route::get('/max', [PostsController::class, 'aggreGatesMax']);
Route::get('/min', [PostsController::class, 'aggreGatesMin']);
Route::get('/avg', [PostsController::class, 'aggreGatesAvg']);
Route::get('/sum', [PostsController::class, 'aggreGatesSum']);
```

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

Answer:- In Laravel's query builder, the `whereNot()` method is used to add a "where not" condition to a query. It allows you to retrieve records that do not match a certain condition. The `whereNot()` method is used to negate the condition specified in the `where()` method.

```php
//Answer of question no 12
public function postWhereNot() {
    $users = DB::table('products')
        ->whereNot('price','=',2000)
        ->get();
    return $users;

}
```

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?

Answer:- The `exists()` and `doesntExist()` methods are used to check the existence of records in a database table.

`exists()`: The `exists()` method is used to check if any records exist in a table that match a given condition. It returns a boolean value of `true` if at least one record exists; otherwise, it returns `false`.

`doesntExist()`: The `doesntExist()` method is the negation of `exists()`. It checks if no records exist in a table that match a given condition. It returns `true` if no matching records are found; otherwise, it returns `false`.

```php
//Answer of question no 13
public function postExists() {
    $hasPosts = DB::table('posts')
        ->where('slug', 'Lorem')
        ->exists();
    return $hasPosts;

}
public function postDoesntExist() {
    $hasPosts = DB::table('posts')
        ->where('slug', 'news')
        ->doesntExist();
    return $hasPosts;

}
```

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.

```php
//Answer of question no 13
public function postmin_to_read() {
    $posts = DB::table('posts')
        ->whereBetween('min_to_read', [1, 5])
        ->get();
    return $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.

```php
//Answer of question no 13
public function postmintoRead() {
    $posts = DB::table('posts')
        ->where('id', 3)
        ->increment('min_to_read');
    return $posts;
}
```