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

Soln: In Laravel the database query builder provides an easy interface to create and run database queries. It can be used to perform all the database operations in our application, from basic database Connection, CRUD, Aggregates, etc. and it works on all supported database systems.

The notable factor about query builder is that, since it uses the PHP Data Objects (PDO), we need not worry about SQL injection attacks. We can avoid all those lines of code to sanitize the data before feeding it to the database.

The DB class is a facade that provides a convenient and expressive interface to interact with the database using the Laravel Query Builder. It allows us to perform database operations such as querying data, inserting records, updating records, deleting records, and executing raw SQL statements.

We create a simple select query to fetch all values from the students table using query builder in the following way,

```
$students = DB::table('students')->get();
```

DB::table is responsible to begin a fluent query against a database table. The table from which the value has to be selected is mentioned inside the brackets within quotes and finally the get() method gets the values. Similarly to fetch a single row we can modify the above code by adding a where clause in the following manner,

```
$students = DB::table('students')->where('name', 'Anik')->first();
```

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.

Soln:

```
class demoController extends Controller
{
    function question2(Request $r)
    {
        $posts= DB::table('posts')->select('excerpt','description')->get();
        print_r($posts->toArray());

    }
}
```

3. Describe the purpose of the distinct( ) method in Laravel's query builder. How is it used in conjunction with the select() method?

Soln: The distinct( ) method is used to fetch distinct records from the database, it is also part of Laravel query builder, which means it can be chained to other query builder methods as well. This method can be used on a posts data table to fetch distinct records in the following manner,

```
class demoController extends Controller
{
    function question3(Request $r)
    {
        $user_ids= DB::table('posts')->distinct()->get();
        print_r($user_ids->toArray());

    }}
```

The select() method can be used along distinct() method . For example we can get all the distinct user_id from the posts table by chaining the select() and distinct() methods. First we select all the user_id from the posts table then we use distinct method to take the unique user_id's.

```
class demoController extends Controller
{
    function question3(Request $r)
    {
        $user_ids= DB::table('posts')->select('user_id')->distinct()->get();
        print_r($user_ids->toArray());
    }
}
```

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.

Soln:

```
class demoController extends Controller
{
    function question4(Request $r)
    {
        $posts= DB::table('posts')->where('id','=',2)->first();
        echo $posts->description;
    }
}
```

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.

Soln:

```
class demoController extends Controller
{
    function question5(Request $r)
    {
   $posts= DB::table('posts')->select('description')->where('id','=',2)->get();
        print_r($posts->toArray());
    }
}
```

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

Soln:

first()	**find()**
1.The **first()** method retrieves the first record that matches the query criteria.	The **find()** method retrieves a record by its primary key value.
2. It returns a single model instance or a null value if no matching record is found.	It returns a single model instance if a matching record is found, and it returns null if no record with the specified primary key is found
3. It is commonly used when you expect a single record to be returned or when you want to retrieve the first record that satisfies your query conditions.	It is typically used when you want to retrieve a record based on its unique identifier (primary key).

The **find()** method can be used on a posts table to find a post with a perticular id in the following manner,

```
$post= DB::table('posts')->find(2);
```

This query returns a single record with id value of 2.
Similary we can use **first()** method on a posts table to retrieve the first record that has a particular user_id in the following manner,

```
$posts= DB::table('posts')->where('user_id','=',1)->first();
```

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.
Soln:

```
class demoController extends Controller
{
    function question7(Request $r)
    {
        $posts= DB::table('posts')->pluck('title');
        print_r($posts->toArray());
    }
}
```

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.

Soln:

```
class demoController extends Controller
{
    function question8(Request $r)
    {
        $result= DB::table('posts')->insert(
            ['user_id'=>2,'title'=>'X','slug'=>'X',
            'excerpt'=>'excerpt','description'=>'description',
            'is_published'=>true,'min_to_read'=>2]);
        echo $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.

Soln:

```
class demoController extends Controller
{
    function question9(Request $r)
    {
        $result= DB::table('posts')->where('id',2)->update(
        ['excerpt'=>'Laravel 10','description'=>'Laravel 10']);
        echo "Number of affected rows ".$result;
    }
}
```

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.

Soln:

```
class demoController extends Controller
{
    function question10(Request $r)
    {
        $result= DB::table('posts')->where('id',3)->delete();
        echo "Number of affected rows ".$result;
    }
}
```

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.

Soln:

Laravel **count()** aggregate function use for getting number of rows in a table or number of records that match a given condition. For example we can find the number of employees whose salary is grater than 6000.

```
  $count = DB::table('employees')->where('salary', '>', '6000')->count();
```

Laravel **max()** aggregate function use for calculate maximum or largest value of numbers. Here's an example to find highest price in products table.

```
 $max_price = DB::table('products')->max('price');
```

Laravel **min()** aggregate function use for calculate minimum or smallest value of numbers. Here's an example to find lowest price in products table.

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

The **sum()** aggregate function it calculate addition of one table column of given rows. This addition can be done according to given condition. For example we can find sum of salary of employees with manager role.

```
$totalSalary = DB::table('employees')->where('role','manager')->sum('salary');
```

To calculate average value of some numbers we use **avg()** aggregate function. For example we can calculate average salary of managers in employee table.

```
$averageSalary= DB::table('employees')->where('role','manager')->avg('salary');
```

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

Soln:

The **whereNot()** method may be used to negate a given group of query constraints. For example, the following query excludes products that are discounted or which have a price that is less than 1000:

```
$products = DB::table('products')->whereNot(function (Builder $query) {
                   $query->where('discounted', true)
                         ->orWhere('price', '<', 1000)
                          })->get();
```

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?

Soln:

The **exists()** method is used to determine if any records exist in the table that match the query conditions. It returns a boolean value indicating whether the query resulted in any matching records. If at least one record is found, **exists()** will return true; otherwise, it will return false. Here's an example to check if any employee with salary over 50000 exists in employee table

```
if (DB::table('employees')->where('salary','>', 50000)->exists()) {
    // Records exist
} else {
    // No records found
}
```

The **doesntExist()** method is the opposite of **exists().** It checks if no records exist in the table that match the query conditions. If no matching records are found, **doesntExist()** returns true; otherwise, it returns false. Here's an example to check if any employee with salary less than 10000 exists in employee table,

```
if (DB::table('employees')->where('salary','<', 10000)->doesntExist()) {
    // No records found
} else {
    // Records exist
}
```

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.

Soln:

```
class demoController extends Controller
{
    function question14(Request $r)
    {
        $posts= DB::table('posts')->whereBetween('min_to_read',[1,5])->get();
        print_r($posts->toArray());
    }
}
```

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.

Soln:

```
class demoController extends Controller
{
    function question15(Request $r)
    {
        $result= DB::table('posts')->where('id',3)->increment('min_to_read',1);
        echo "Number of affected rows ".$result;
    }
}
```