1.

Laravel's query builder is a feature of the Laravel framework that provides a simple and elegant way to interact with databases. It allows developers to build and execute database queries using a fluent, expressive syntax.

The query builder is essentially an object-oriented interface to construct SQL queries programmatically. It abstracts the underlying database-specific SQL syntax and provides a more readable and intuitive approach to building queries. Instead of writing raw SQL statements, you can use methods and chain them together to construct queries in a more human-readable manner.

Here are some key features and benefits of Laravel's query builder:

- Fluent interface: The query builder uses a fluent interface, which means you can chain methods together to construct complex queries in a readable and concise manner. For example, you can start with a base query and then chain methods like select, where, orderBy, and so on to build the desired query.

- Parameter binding: The query builder automatically handles parameter binding, which helps prevent SQL injection attacks. Instead of directly concatenating user input into the query, you can use placeholders and pass the values as parameters, ensuring that the input is properly escaped.

- Database agnostic: The query builder is designed to work with multiple database systems, including MySQL, PostgreSQL, SQLite, and SQL Server. It abstracts the differences between these databases, allowing you to write portable queries that can be executed across different database platforms.

- Query building shortcuts: Laravel's query builder provides shortcuts for common SQL operations. For example, you can use methods like insert, update, and delete to perform corresponding operations without writing the full SQL statements explicitly. These shortcuts simplify the code and make it more readable.

- Eloquent integration: Laravel's query builder is seamlessly integrated with the Eloquent ORM (Object-Relational Mapping) in Laravel. Eloquent provides an active record implementation, and the query builder allows you to construct queries for Eloquent models. This integration enables you to combine the power of the query builder with the convenience of working with models and relationships.

Overall, Laravel's query builder provides a powerful and expressive way to interact with databases. It allows developers to write database queries in a clean and readable syntax, abstracts the underlying database details, and integrates well with other components of the Laravel framework. These features contribute to a simpler and more elegant database interaction experience for Laravel developers.

2.

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

3.

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

When used in conjunction with the select() method, distinct() modifies the SELECT clause of the query to include the DISTINCT keyword. This tells the database to return only unique values for the specified columns.

4.

```
// $posts = DB::table('posts')->find(1002);

$posts = DB::table('posts')->where('id',1002)->first();
return $posts->description;
```

5.

```
$posts = DB::table('posts')->select('description')
            ->where('id',1002)
            ->first();
return $posts;
```

6.

In Laravel's query builder, the first() and find() methods are used to retrieve single records from a database table. However, they have slight differences in terms of their usage and behavior.

- first(): The first() method is used to retrieve the first record that matches the query conditions. It returns a single instance of the model or a plain PHP object representing the first row in the result set. If the query does not match any records, null is returned.
- find(): The find() method is used to retrieve a record by its primary key. It expects the primary key value as an argument and returns a single instance of the model or a plain PHP object representing the record. If the record with the specified primary key does not exist, null is returned.

7.

```
    $posts = DB::table('posts')->select('title')->get();
    return $posts;
```

8.

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

9.

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

10.

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

11.

In Laravel's query builder, aggregate methods such as count(), sum(), avg(), max(), and min() are used to perform calculations on a specific column of a database table. These methods allow you to retrieve aggregated values based on the query conditions.

Here's an explanation of each aggregate method along with an example:

- count(): The count() method is used to retrieve the number of records that match the query conditions. It returns the count as an integer value.

```
$count = DB::table('posts')->where('is_published', true)->count();

return $count;
```

- sum(): The sum() method calculates the sum of a specific column in the matched records. It returns the sum as a numeric value.

```
$total = DB::table('posts')->sum('min_to_read');
return $total;
```

- avg(): The avg() method calculates the average (mean) value of a specific column in the matched records. It returns the average as a numeric value.

```
$avg = DB::table('posts')->avg('min_to_read');
return $avg;
```

- max(): The max() method retrieves the maximum value from a specific column in the matched records. It returns the maximum value.

```
$max = DB::table('posts')->max('min_to_read');
return $max;
```

- min(): The min() method retrieves the minimum value from a specific column in the matched records. It returns the minimum value.

```
$min = DB::table('posts')->min('min_to_read');
return $min;
```

12.

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

```
$posts = DB::table('posts')
      ->whereNot('is_published', true)
      ->get();
return $posts;
```

13.

In Laravel's query builder, the exists() and doesntExist() methods are used to check the existence of records in a database table. However, they have opposite meanings and return different results.

- exists(): The exists() method is used to check if any records match the query conditions. It returns true if at least one record exists that satisfies the query, and false otherwise.

```
$minToRead = DB::table('posts')->where('min_to_read', 0)->exists();
return $minToRead;
```

- doesntExist(): The doesntExist() method is the negation of exists(). It checks if no records match the query conditions. It returns true if no record exists that satisfies the query, and false if there is at least one matching record.

```
$noMinToRead = DB::table('posts')-
                    >where('min_to_read', 10)
                    ->doesntExist();
return $noMinToRead;
```

14.

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

15.

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