Question 1:
Laravel's query builder is a feature of the Laravel framework that provides a fluent, expressive, and intuitive interface for interacting with databases. It allows developers to build and execute database queries using method chaining and a set of convenient methods, resulting in a simpler and more readable code.
Here are some key aspects of Laravel's query builder that contribute to its simplicity and elegance:
1. Fluent and expressive syntax: The query builder utilizes a fluent interface, allowing you to chain methods together to construct queries. This approach leads to more readable and concise code, as the query construction is done in a natural and sequential manner.
2. Database agnostic: Laravel's query builder is designed to work with multiple database systems, including MySQL, PostgreSQL, SQLite, and SQL Server. It abstracts away the differences between these databases, enabling you to write database-agnostic queries without worrying about specific syntax or features.
3. Parameter binding: The query builder automatically handles parameter binding, which helps prevent SQL injection attacks and ensures the proper escaping of user-supplied values. You can pass parameters to the query builder using placeholders, which are automatically bound by

the framework.
4. Query building methods: Laravel's query builder provides a comprehensive set of methods for constructing queries. You can easily specify conditions, joins, ordering, grouping, and other query components using intuitive methods such as where(), join(), orderBy(), groupBy(), and more.
5. Eloquent ORM integration: The query builder is tightly integrated with Laravel's Eloquent ORM (Object-Relational Mapping) system, allowing you to seamlessly switch between using raw queries and working with ORM models. This integration enables you to combine the benefits of both approaches, depending on your specific use case.
6. Result set manipulation: The query builder offers convenient methods for manipulating and transforming result sets. You can use methods like pluck(), count(), max(), min(), avg(), and others to retrieve specific values, aggregate data, or perform calculations on the result set.
Overall, Laravel's query builder provides an elegant way to interact with databases by abstracting the complexities of SQL syntax and database-specific features. It empowers developers to write expressive and maintainable database queries, leading to efficient database operations and streamlined development processes.

Question 2:

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

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

print_r($posts);
```

Question 3:

The distinct() method in Laravel's query builder is used to retrieve only the unique values from a specific column in the result set. It helps in filtering out duplicate values and ensures that each value appears only once in the final result.
When used in conjunction with the select() method, the distinct() method applies the uniqueness constraint to the specified column(s). It ensures that the selected columns contain only unique values, eliminating any duplicates from the result set.
Here's an example to illustrate its usage:
php

```
use Illuminate\Support\Facades\DB;
$uniqueEmails = DB::table('users') ->select('email')
->distinct() ->get();
```

In this example, we are retrieving the unique email addresses from the "users" table. The select('email') method specifies that we are only interested in the "email" column. By chaining the distinct() method after the select() method, we ensure that the resulting emails in $uniqueEmails will be distinct, and any duplicates will be removed. Note that the distinct() method can be used with multiple columns as well. In that case, it ensures the uniqueness of combinations of values in those columns, rather than individual values.

Question 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.";
}
```

Question 5:

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

Question 6:

* first() retrieves the first record matching the query criteria.
* find() retrieves a record by its primary key value.
Both methods are used to retrieve single records, but first() is more flexible in terms of defining query conditions, while find() is specifically used for retrieving records by their primary key value.

Question 7:

```
$posts = DB::table('posts')->pluck('title');

print_r($posts);
```

Question 8:

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

// ...

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

```
$result = DB::table('posts')->orderBy('id', 'desc')-
>first();

print_r($result);
```

Qestion 9:

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

// ...

$id = 2;
$newValues = 'Laravel 10';

$affectedRows = DB::table('posts')
    ->where('id', $id)
    ->update([
        'excerpt' => $newValues,
        'description' => $newValues
    ]);

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

Question 10:

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

// ...

public function deletePost($id)
```

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

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

Question 11:
1. count(): The count() method returns the number of records that match a specific condition. It can be used to count the total number of rows in a table or the number of rows that satisfy certain criteria.
Example:
php

$count = DB::table('users')->count(); echo "Total users: " . $count;
2. sum(): The sum() method calculates the sum of a column's values for the selected records. It is commonly used to find the total of a numeric column, such as the total sales amount.
Example:

$totalSales = DB::table('orders')->sum('amount');
echo "Total sales: " . $totalSales;
3. avg(): The avg() method calculates the average value of a column for the selected records. It is

useful for obtaining the average value of a numeric field, like the average rating of products.
Example:
php
$averageRating = DB::table('products')->avg('rating'); echo "Average rating: " . $averageRating;
4. max(): The max() method retrieves the maximum value of a column from the selected records. It is commonly used to find the highest value in a numeric column, such as the maximum price of a product.

$maxPrice = DB::table('products')->max('price'); echo "Maximum price: " . $maxPrice;
5. min(): The min() method fetches the minimum value of a column from the selected records. It is used to find the lowest value in a numeric column, such as the minimum age of users.

$minAge = DB::table('users')->min('age'); echo "Minimum age: " . $minAge;
These aggregate methods are helpful for performing calculations on large data sets efficiently and obtaining meaningful insights from your database records.

exists(): This method is used to check if any records exist that match the specified conditions. It returns a boolean value of true if at least one record is found, and false otherwise. The exists() method can be chained with other query builder methods to build complex queries.

doesntExist(): This method is the opposite of exists(). It checks if no records exist that match the specified conditions. It returns a boolean value of true if no records are found, and false if there is at least one matching record.

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

// ...

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

print_r($posts);
```

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

// ...
```

```
$id = 3;

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

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