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

Laravel's query builder is a feature of the Laravel framework that provides a simple and elegant way to interact with databases. It offers a fluent interface for constructing and executing database queries in a convenient manner. The query builder allows developers to write database queries using PHP code instead of directly writing SQL statements, which can make the code more readable and maintainable.

With the Laravel query builder, you can perform various operations on the database, including selecting, inserting, updating, and deleting records. It supports all the major database systems that Laravel supports, such as MySQL, PostgreSQL, SQLite, and SQL Server.

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

Fluent Interface: The query builder uses a fluent interface, which means that you can chain methods together to build complex queries in a readable and expressive way. This makes it easy to understand and modify queries as needed.

Parameter Binding: The query builder uses PDO parameter binding to protect against SQL injection attacks. This means that you don't need to manually sanitize or escape user input when using the query builder, as the bindings take care of it for you.

Query Building Methods: Laravel's query builder provides a wide range of methods to construct queries. You can use methods like select, where, orderBy, groupBy, limit, and offset to build queries with different conditions and constraints.

Eloquent Integration: The query builder is seamlessly integrated with Laravel's Eloquent ORM, allowing you to combine the power of both in your applications. You can use the query builder to construct queries and then use Eloquent models to work with the retrieved data in an object-oriented way.

Database Aggregates: The query builder includes methods for calculating aggregate values like count, max, min, avg, and sum. These methods allow you to easily retrieve aggregated information from the database.

Raw Expressions: In some cases, you may need to include raw SQL expressions in your queries. The query builder provides a raw method that allows you to insert raw SQL snippets into your queries while still benefiting from parameter binding.

By providing a high-level abstraction over database operations, Laravel's query builder simplifies the process of interacting with databases and promotes clean and maintainable code. It combines the flexibility and power of SQL with the elegance and expressiveness of PHP, making database operations in Laravel a breeze.

Question- 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.

Route:

```
Route::get( '/excerptDescription', [PostController::class,
'getExcerptDescription'] );
```

PostController:

```
public function getExcerptDescription() {
  $posts = DB::table( 'posts' )->select( 'excerpt', 'description' )->get();
  return $posts;
 }
```

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

In Laravel's query builder, the distinct() method is used to retrieve **unique records** from a database table. It only returns the distinct rows by eliminating all duplicates.

When used in conjunction with the select() method, the distinct() method operates on the columns specified in the select() method. It modifies the query to return only unique values for the selected columns. This can be useful when you want to retrieve a list of unique values for specific columns in a table.

```
Route::get( '/distinct', [PostController::class, 'distinct'] );
```

```
public function distinct() {
  $posts = DB::table( 'posts' )->select( 'author' )->distinct()->get();
  return $posts;
 }
```

Question- 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.

```
Route::get( '/firstDes', [PostController::class, 'firstDes'] );
```

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

Question- 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.

```
Route::get( '/getDescription', [PostController::class, 'getDescription'] );
```

```php
public function getDescription() {
  $posts = DB::table( 'posts' )->where( 'id', 2 )->pluck( 'description' );
  return $posts;
 }
```

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

Both the first() and the find() methods are used to retrieve single records from a table of a database. However, they have different behaviors and are used in different contexts.

The first() method is used to retrieve the first record that matches the query criteria. It returns a single instance of the model or null if no matching record is found. The first() method is typically used when you want to retrieve the earliest record based on a specific ordering.

```php
$user = DB::table('users')->first();
```

The find() method is used to retrieve a record by its primary key. It expects the primary key value as its argument and returns a single instance of the model if a record is found, or null if no record matches the primary key value.

```php
$user = DB::table('users')->find(1);
```

Question- 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.

```php
Route::get( '/pluck', [PostController::class, 'pluck'] );
```

```php
 function pluck() {
  $posts = DB::table( 'posts' )->pluck( 'title' );
  return $posts;
 }
```

Question- 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.

```
Route::post( '/insertData', [PostController::class, 'insertData'] );

 public function insertData() {
  DB::table( 'posts' )->insert( [
   'title'         => 'Custom Post',
   'slug'          => 'custom-post',
   'description'   => 'description',
   'excerpt'       => 'excerpt',
   'is_published' => true,
   'min_to_read'   => 2,
  ] );

  // Print the result of the insert operation
  return 'New record inserted successfully!';
 }
```

Question- 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.

```
Route::put( '/posts/{id}', [PostController::class, 'update'] );
```

```
public function update() {
    $id = 2;
$newValues = 'Laravel 10';

$affectedRows = DB::table('posts')
    ->where('id', $id)
    ->update([
        'excerpt' => $newValues,
        'description' => $newValues
    ]);
return "Number of affected rows: " . $affectedRows;
}
```

Question- 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.

```
Route::delete( '/delPost', [PostController::class, 'destroy'] );
```

```
 function destroy() {
  $deletedRows = DB::table( 'posts' )->where( 'id', '=', 3 )->delete();
  return "Number of affected rows: " . $deletedRows;
 }
```

Question- 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.

```
Route::get( '/question11', [PostController::class, 'countPrice'] );
```

```
public function countPrice() {

  $count = DB::table( 'users' )->count();
  $count = "Total number of users: " . $count;

  $totalPrice = DB::table( 'products' )->sum( 'price' );
  $totalPrice = "Total price of all products: " . $totalPrice;

  $avgPrice = DB::table( 'products' )->avg( 'price' );
  $avgPrice = "Average price of all products: " . $avgPrice;

  $maxPrice = DB::table( 'products' )->max( 'price' );
  $maxPrice = "Maximum price of a products: " . $maxPrice;

  $minPrice = DB::table( 'products' )->min( 'price' );
  $minPrice = "Minimum price of a product: " . $minPrice;

  return response()->json(
   [
    'coutn'       => $count,
    'totalPrice' => $totalPrice,
    'avgPrice'    => $avgPrice,
    'maxPrice'    => $maxPrice,
    'minPrice'    => $minPrice,
   ] );
 }
```

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

The whereNot() method in Laravel's query builder is used to add a "not" condition to a query. It allows you to retrieve records where a particular column's value is not equal to the specified value.

Here's an example of how the whereNot() method can be used:

```
Route::get( '/question12', [PostController::class, 'question12'] );
```

```
public function question12() {
 $posts = DB::table( 'posts' )->whereNot( 'user_id', 2 )->get();
 return $posts;
}
```

Question- 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?

The exists() and doesntExist() methods in Laravel's query builder are used to check the existence of records in a database table based on specific conditions.

The exists() method is used to determine if there are any records in the table that match the given query constraints. It returns true if at least one record exists; otherwise, it returns false. Here's an example of how exists() can be used:

```
$hasPost = DB::table('users')->where('name', 'John')->exists();
```

In the example above, the exists() method checks if there is at least one record in the "users" table where the "status" column is set to "active".

On the other hand, the doesntExist() method is the opposite of exists(). It returns true if no records exist that match the given query constraints; otherwise, it returns false. Here's an example:

```
$hasNoPosts = DB::table('users')->where('name', 'John')->doesntExist();
```

In this example, the doesntExist() method checks if there are no records in the "users" table with a "status" column set to "active".

Both exists() and doesntExist() methods are useful for conditional logic based on the presence or absence of records in the database. They can be chained with other query builder methods to specify more specific conditions.

Question- 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.

```
Route::get( '/whereBetween', [PostController::class, 'whereBetween'] );
```

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

Question- 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.

```
Route::get( '/increment', [PostController::class, 'increment'] );
```

```
public function increment() {
  $affectedRows = DB::table( 'posts' )
   ->where( 'id', 3 )
   ->increment( 'min_to_read', 1 );

  return "Number of affected rows: " . $affectedRows;
 }
```