1. Laravel's query builder is a fluent interface that provides a simple and elegant way to interact with databases. It allows developers to build database queries using chainable methods, providing a more readable and maintainable syntax compared to writing raw SQL queries. The query builder supports various database operations such as selecting, inserting, updating, and deleting records. It also supports advanced features like joining tables, subqueries, and parameter binding, making it a powerful tool for working with databases in Laravel applications.

2. Code to retrieve the "excerpt" and "description" columns from the "posts" table using Laravel's query builder:

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

print_r($posts);
```

3. The `distinct()` method in Laravel's query builder is used to retrieve only unique values from a column. It ensures that the query result does not contain duplicate values. It is often used in conjunction with the `select()` method to specify the columns to retrieve, and the distinct values are applied to those selected columns.

4. Code to retrieve the first record from the "posts" table where the "id" is 2 using Laravel's query builder:

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

echo $posts->description;
```

5. Code to retrieve the "description" column from the "posts" table where the "id" is 2 using Laravel's query builder:

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

echo $posts;
```

6. The `first()` method is used to retrieve the first record matching the query conditions from the database. It returns a single object representing the record. On the other hand, the `find()` method is used to retrieve a record by its primary key. It specifically targets the primary key column and returns a single object. The `first()` method is typically used when querying with conditions other than the primary key, while `find()` is used when searching for a record by its primary key.

7. Code to retrieve the "title" column from the "posts" table using Laravel's query builder:

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

8. Code to insert a new record into the "posts" table using Laravel's query builder:

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

9. 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:

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

10. Code to delete the record with the "id" of 3 from the "posts" table using Laravel's query builder:

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

11. Aggregate methods in Laravel's query builder are used to perform calculations on selected columns or entire tables. Here are the explanations and examples of each aggregate method:

- `count()`: Returns the number of records in a table or the number of selected records based on the query conditions. Example:

```
$count = DB::table('posts')->count();
echo $count;
```

- `sum()`: Calculates the sum of a column's values. Example:

```
$sum = DB::table('orders')->sum('total_amount');
echo $sum;
```

- `avg()`: Calculates the average value of a column. Example:

```
$average = DB::table('products')->avg('price');
echo $average;
```

- `max()`: Retrieves the maximum value of a column. Example:

```
$maxValue = DB::table('scores')->max('score');
echo $maxValue;
```

- `min()`: Retrieves the minimum value of a column. Example:

```
$minValue = DB::table('temperatures')->min('value');
echo $minValue;
```

12. The `whereNot()` method in Laravel's query builder is used to specify a condition where a column's value is not equal to a given value. It excludes records that match the specified condition from the query result. Here's an example usage:

```
$posts = DB::table('posts')
    ->whereNot('category', 'News')
    ->get();
print_r($posts);
```

This code retrieves all records from the "posts" table where the "category" is not equal to 'News'. The `whereNot()` method excludes records with the category 'News' from the query result.

13. The `exists()` and `doesntExist()` methods are used in Laravel's query builder to check the existence of records.

- `exists()`: Returns true if any records match the query conditions; otherwise, returns false. Example:

```
$exists = DB::table('users')->where('email', 'example@example.com')->exists();
echo $exists ? 'Record exists' : 'Record does not exist';
```

- `doesntExist()`: Returns true if no records match the query conditions; otherwise, returns false. Example:

```
$doesntExist = DB::table('users')->where('email', 'example@example.com')->doesntExist();
echo $doesntExist ? 'Record does not exist' : 'Record exists';
```

14. Code to retrieve records from the "posts" table where the "min_to_read" column is between 1 and 5 using Laravel's query builder:

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

This code retrieves all records from the "posts" table where the "min_to_read" column is between 1 and 5, inclusive.

15. 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:

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

This code increments the "min_to_read" column value of the record with the ID of 3 by 1. The `increment()` method automatically updates the column value and returns the number of affected rows, which

in this case would be 1.