1. Laravel's query builder is a fluent and expressive interface for building and executing database queries. It provides a simple and elegant way to interact with databases by allowing developers to write database queries using PHP instead of raw SQL. The query builder offers a wide range of methods for selecting, inserting, updating, and deleting records, as well as joining tables, applying conditions, and ordering results. It abstracts the underlying database engine, making it easy to switch between different databases without changing the code. Laravel's query builder promotes code readability and helps prevent SQL injection attacks by automatically sanitising user inputs.

2. Code to retrieve "excerpt" and "description" columns from the "posts" table and store the result in $posts variable:

```php
$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 the unique values of a specific column from the result set. It eliminates duplicate rows and returns a distinct set of records based on the specified column. The distinct() method is used in conjunction with the select() method to indicate which column's unique values should be retrieved.

4. Code to retrieve the first record from the "posts" table where the "id" is 2 and print the "description" column:

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


if ($posts) {
    echo $posts->description;
}
```

5. Code to retrieve the "description" column from the "posts" table where the "id" is 2 and store the result in the $posts variable:

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


echo $posts;
```

6. The first() method in Laravel's query builder is used to retrieve the first record that matches the query conditions. It returns a single object representing the first row of the result set. On the other hand, the find() method retrieves a single record by its primary key. It specifically searches for a record with the given primary key value. If the record is found, it returns an object representing that record; otherwise, it returns null.

7. Code to retrieve the "title" column from the "posts" table and store the result in the $posts variable:

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


print_r($posts);
```

8. Code to insert a new record into the "posts" table:

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

```
print_r($result);
```

9. Code to update the "excerpt" and "description" columns of the record with the "id" of 2 in the "posts" table:

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


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

10. Code to delete the record with the "id" of 3 from the "posts" table:

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


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

11. Laravel's query builder provides several aggregate methods for performing calculations on the data in the result set:

count(): Returns the number of records in the result set. Example: DB::table('posts')->count().
sum(): Returns the sum of a specific column in the result set. Example: DB::table('orders')->sum('amount').
avg(): Returns the average value of a specific column in the result set. Example: DB::table('products')->avg('price').
max(): Returns the maximum value of a specific column in the result set. Example: DB::table('users')->max('age').

min(): Returns the minimum value of a specific column in the result set. Example: DB::table('products')→min('stock').

12. The whereNot() method in Laravel's query builder is used to add a "not equals" condition to a query. It retrieves records where a specific column's value is not equal to the given value. Here's an example:

```
$posts = DB::table('posts')
            →whereNot('status', 'published')
            →get();
```

13. The exists() method is used to check if any records exist in the query result. It returns true if there is at least one record; otherwise, it returns false. On the other hand, the doesntExist() method is the negation of exists(). It returns true if no records exist in the query result; otherwise, it returns false.

```
$exists = DB::table('posts')
            →where('status', 'published')
            →exists();


$doesntExist = DB::table('posts')
                →where('status', 'published')
                →doesntExist();
```

14. Code to retrieve records from the "posts" table where the "min_to_read" column is between 1 and 5:

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

print_r($posts);
```

15. Code to increment the "min_to_read" column value of the record with the "id" of 3 in the "posts" table by 1:

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


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