**Laravel Query Builder Cheat Sheet**

#### [***Retrieving A Single Row / Column From A Table***](https://laravel.com/docs/8.x/queries#retrieving-a-single-row-column-from-a-table)

**$user = DB::table('users')->where('name', 'John')->first();**

**If you don't need an entire row, you may extract a single value from a record using the value method. This method will return the value of the column directly:**

**$email = DB::table('users')->where('name', 'John')->value('email');**

**To retrieve a single row by its id column value, use the find method:**

**$user = DB::table('users')->find(3);**

#### [***Retrieving A List Of Column Values***](https://laravel.com/docs/8.x/queries#retrieving-a-list-of-column-values)

**If you don't need an entire row, you may extract a single value from a record using the value method. This method will return the value of the column directly:**

$titles = DB::table('users')->pluck('title');

### [Chunking Results](https://laravel.com/docs/8.x/queries#chunking-results)

This method retrieves a small chunk of results at a time and feeds each chunk into a closure for processing. For example, let's retrieve the entire users table in chunks of 100 records at a time

**DB::table('users')->orderBy('id')->chunk(100, function ($users) {**

**foreach ($users as $user) {**

**//**

**}**

**});**

### [Aggregates](https://laravel.com/docs/8.x/queries#aggregates)

**$users = DB::table('users')->count();**

**$price = DB::table('orders')->max('price');**

**Of course, you may combine these methods with other clauses to fine-tune how your aggregate value is calculated:**

**$price = DB::table('orders')**

**->where('finalized', 1)**

**->avg('price');**

#### [**Determining If Records Exist**](https://laravel.com/docs/8.x/queries#determining-if-records-exist)

if (DB::table('orders')->where('finalized', 1)->exists()) {

// ...

}

if (DB::table('orders')->where('finalized', 1)->doesntExist()) {

// ...

}

## [Select Statements](https://laravel.com/docs/8.x/queries#select-statements)

#### [Specifying A Select Clause](https://laravel.com/docs/8.x/queries#specifying-a-select-clause)

$users = DB::table('users')

->select('name', 'email as user\_email')

->get();

**The distinct method allows you to force the query to return distinct results:**

**$users = DB::table('users')->distinct()->get()**

**If you already have a query builder instance and you wish to add a column to its existing select clause, you may use the addSelect method:**

**$query = DB::table('users')->select('name');**

**$users = $query->addSelect('age')->get();**

## [Raw Expressions](https://laravel.com/docs/8.x/queries#raw-expressions)

**To create a raw string expression, you may use the raw method provided by the DB facade:**

**$users = DB::table('users')**

**->select(DB::raw('count(\*) as user\_count, status'))**

**->where('status', '<>', 1)**

**->groupBy('status')**

**->get();**

### [Raw Methods](https://laravel.com/docs/8.x/queries#raw-methods)

#### [**selectRaw**](https://laravel.com/docs/8.x/queries#selectraw)

$orders = DB::table('orders')

->selectRaw('price \* ? as price\_with\_tax', [1.0825])

**->get();**

#### [**whereRaw / orWhereRaw**](https://laravel.com/docs/8.x/queries#whereraw-orwhereraw)

$orders = DB::table('orders')

->whereRaw('price > IF(state = "TX", ?, 100)', [200])

->get();

#### [**havingRaw / orHavingRaw**](https://laravel.com/docs/8.x/queries#havingraw-orhavingraw)

$orders = DB::table('orders')

->select('department', DB::raw('SUM(price) as total\_sales'))

->groupBy('department')

->havingRaw('SUM(price) > ?', [2500])

->get();

#### [**orderByRaw**](https://laravel.com/docs/8.x/queries#orderbyraw)

$orders = DB::table('orders')

->orderByRaw('updated\_at - created\_at DESC')

->get();

### [**groupByRaw**](https://laravel.com/docs/8.x/queries#groupbyraw)

$orders = DB::table('orders')

->select('city', 'state')

->groupByRaw('city, state')

->get();

## [Joins](https://laravel.com/docs/8.x/queries#joins)

#### [**Inner Join Clause**](https://laravel.com/docs/8.x/queries#inner-join-clause)

$users = DB::table('users')

->join('contacts', 'users.id', '=', 'contacts.user\_id')

->join('orders', 'users.id', '=', 'orders.user\_id')

->select('users.\*', 'contacts.phone', 'orders.price')

->get();

#### [Left Join / Right Join Clause](https://laravel.com/docs/8.x/queries#left-join-right-join-clause)

$users = DB::table('users')

->leftJoin('posts', 'users.id', '=', 'posts.user\_id')

->get();

$users = DB::table('users')

->rightJoin('posts', 'users.id', '=', 'posts.user\_id')

->get();

#### [Cross Join Clause](https://laravel.com/docs/8.x/queries#cross-join-clause)

$sizes = DB::table('sizes')

->crossJoin('colors')

->get();

#### [***Advanced Join Clauses***](https://laravel.com/docs/8.x/queries#advanced-join-clauses)

DB::table('users')

->join('contacts', function ($join) {

$join->on('users.id', '=', 'contacts.user\_id')->orOn(...);

})

->get();

If you would like to use a "where" clause on your joins, you may use the where and orWhere methods provided by the JoinClause instance.

DB::table('users')

->join('contacts', function ($join) {

$join->on('users.id', '=', 'contacts.user\_id')

->where('contacts.user\_id', '>', 5);

})

**->get();**

#### [***Subquery Joins***](https://laravel.com/docs/8.x/queries#subquery-joins)

You may use the joinSub, leftJoinSub, and rightJoinSub methods to join a query to a subquery

->select('user\_id', DB::raw('MAX(created\_at) as last\_post\_created\_at'))

->where('is\_published', true)

->groupBy('user\_id');

$users = DB::table('users')

->joinSub($latestPosts, 'latest\_posts', function ($join) {

$join->on('users.id', '=', 'latest\_posts.user\_id');

})->get();

## [Unions](https://laravel.com/docs/8.x/queries#unions)

The query builder also provides a convenient method to "union" two or more queries together. For example, you may create an initial query and use the union method to union it with more queries:

use Illuminate\Support\Facades\DB;

$first = DB::table('users')

->whereNull('first\_name');

$users = DB::table('users')

->whereNull('last\_name')

->union($first)

->get();

## [Basic Where Clauses](https://laravel.com/docs/8.x/queries#basic-where-clauses)

### [Where Clauses](https://laravel.com/docs/8.x/queries#where-clauses)

$users = DB::table('users')

->where('votes', '=', 100)

->where('age', '>', 35)

->get();

$users = DB::table('users')->where('votes', 100)->get();

you may use any operator that is supported by your database system:

$users = DB::table('users')

->where('votes', '>=', 100)

->get();

$users = DB::table('users')

->where('votes', '<>', 100)

->get();

$users = DB::table('users')

->where('name', 'like', 'T%')

->get();

You may also pass an array of conditions to the where function.

$users = DB::table('users')->where([

['status', '=', '1'],

['subscribed', '<>', '1'],

])->get();

### [Or Where Clauses](https://laravel.com/docs/8.x/queries#or-where-clauses)

you may use the orWhere method to join a clause to the query using the or operator. The orWhere method accepts the same arguments as the where method:

$users = DB::table('users')

->where('votes', '>', 100)

->orWhere('name', 'John')

->get();

If you need to group an "or" condition within parentheses, you may pass a closure as the first argument to the orWhere method:

$users = DB::table('users')

->where('votes', '>', 100)

->orWhere(function($query) {

$query->where('name', 'Abigail')

->where('votes', '>', 50);

})

->get();

whereBetween / orWhereBetween

The whereBetween method verifies that a column's value is between two values:

$users = DB::table('users')

->whereBetween('votes', [1, 100])

->get();

whereNotBetween / orWhereNotBetween

The whereNotBetween method verifies that a column's value lies outside of two values:

$users = DB::table('users')

->whereNotBetween('votes', [1, 100])

->get();

whereIn / whereNotIn / orWhereIn / orWhereNotIn

The whereIn method verifies that a given column's value is contained within the given array:

$users = DB::table('users')

->whereIn('id', [1, 2, 3])

->get();

The whereNotIn method verifies that the given column's value is not contained in the given array:

$users = DB::table('users')

->whereNotIn('id', [1, 2, 3])

->get();

whereNull / whereNotNull / orWhereNull / orWhereNotNull

The whereNull method verifies that the value of the given column is NULL:

$users = DB::table('users')

->whereNull('updated\_at')

->get();

The whereNotNull method verifies that the column's value is not NULL:

$users = DB::table('users')

->whereNotNull('updated\_at')

->get();

whereDate / whereMonth / whereDay / whereYear / whereTime

The whereDate method may be used to compare a column's value against a date:

$users = DB::table('users')

->whereDate('created\_at', '2016-12-31')

->get();

The whereMonth method may be used to compare a column's value against a specific month:

$users = DB::table('users')

->whereMonth('created\_at', '12')

->get();

The whereDay method may be used to compare a column's value against a specific day of the month:

$users = DB::table('users')

->whereDay('created\_at', '31')

->get();

The whereYear method may be used to compare a column's value against a specific year:

$users = DB::table('users')

->whereYear('created\_at', '2016')

->get();

The whereTime method may be used to compare a column's value against a specific time:

$users = DB::table('users')

->whereTime('created\_at', '=', '11:20:45')

->get();

whereColumn / orWhereColumn

The whereColumn method may be used to verify that two columns are equal:

$users = DB::table('users')

->whereColumn('first\_name', 'last\_name')

->get();

You may also pass a comparison operator to the whereColumn method:

$users = DB::table('users')

->whereColumn('updated\_at', '>', 'created\_at')

->get();

You may also pass an array of column comparisons to the whereColumn method. These conditions will be joined using the and operator:

$users = DB::table('users')

->whereColumn([

['first\_name', '=', 'last\_name'],

['updated\_at', '>', 'created\_at'],

])->get();

### [Logical Grouping](https://laravel.com/docs/8.x/queries#logical-grouping)

Sometimes you may need to group several "where" clauses within parentheses in order to achieve your query's desired logical grouping.

$users = DB::table('users')

->where('name', '=', 'John')

->where(function ($query) {

$query->where('votes', '>', 100)

->orWhere('title', '=', 'Admin');

})

->get();

### [Where Exists Clauses](https://laravel.com/docs/8.x/queries#where-exists-clauses)

$users = DB::table('users')

->whereExists(function ($query) {

$query->select(DB::raw(1))

->from('orders')

->whereColumn('orders.user\_id', 'users.id');

})

->get();

The query above will produce the following SQL:

select \* from users

where exists (

select 1

from orders

where orders.user\_id = users.id

)

## [Ordering, Grouping, Limit & Offset](https://laravel.com/docs/8.x/queries#ordering-grouping-limit-and-offset)

### [Ordering](https://laravel.com/docs/8.x/queries#ordering)

#### [The orderBy Method](https://laravel.com/docs/8.x/queries#orderby)

The orderBy method allows you to sort the results of the query by a given column.

$users = DB::table('users')

->orderBy('name', 'desc')

->get();

To sort by multiple columns, you may simply invoke orderBy as many times as necessary:

$users = DB::table('users')

->orderBy('name', 'desc')

->orderBy(

#### [The latest & oldest Methods](https://laravel.com/docs/8.x/queries#latest-oldest)

The latest and oldest methods allow you to easily order results by date.

$user = DB::table('users')

->latest()

->first();

#### [Random Ordering](https://laravel.com/docs/8.x/queries#random-ordering)

$randomUser = DB::table('users')

->inRandomOrder()

->first();

#### [Removing Existing Orderings](https://laravel.com/docs/8.x/queries#removing-existing-orderings)

The reorder method removes all of the "order by" clauses that have previously been applied to the query:

$query = DB::table('users')->orderBy('name');

$unorderedUsers = $query->reorder()->get();

### [Grouping](https://laravel.com/docs/8.x/queries#grouping)

#### [The groupBy & having Methods](https://laravel.com/docs/8.x/queries#groupby-having)

$users = DB::table('users')

->groupBy('account\_id')

->having('account\_id', '>', 100)

->get();

### [Limit & Offset](https://laravel.com/docs/8.x/queries#limit-and-offset)

#### [The skip & take Methods](https://laravel.com/docs/8.x/queries#skip-take)

You may use the skip and take methods to limit the number of results returned from the query or to skip a given number of results in the query:

$users = DB::table('users')->skip(10)->take(5)->get();

Alternatively, you may use the limit and offset methods. These methods are functionally equivalent to the take and skip methods, respectively:

$users = DB::table('users')

->offset(10)

->limit(5)

->get();

## [Insert Statements](https://laravel.com/docs/8.x/queries#insert-statements)

The insert method accepts an array of column names and values:

DB::table('users')->insert([

'email' => 'kayla@example.com',

'votes' => 0

]);

DB::table('users')->insert([

['email' => 'picard@example.com', 'votes' => 0],

['email' => 'janeway@example.com', 'votes' => 0],

]);

The insertOrIgnore method will ignore duplicate record errors while inserting records into the database:

DB::table('users')->insertOrIgnore([

['id' => 1, 'email' => 'sisko@example.com'],

['id' => 2, 'email' => 'archer@example.com'],

]);

#### [Auto-Incrementing IDs](https://laravel.com/docs/8.x/queries#auto-incrementing-ids)

$id = DB::table('users')->insertGetId(

['email' => 'john@example.com', 'votes' => 0]

);

### [Upserts](https://laravel.com/docs/8.x/queries#upserts)

The upsert method will insert records that do not exist and update the records that already exist with new values that you may specify.

DB::table('flights')->upsert([

['departure' => 'Oakland', 'destination' => 'San Diego', 'price' => 99],

['departure' => 'Chicago', 'destination' => 'New York', 'price' => 150]

], ['departure', 'destination'], ['price']);

In the example above, Laravel will attempt to insert two records. If a record already exists with the same departure and destination column values, Laravel will update that record's price column.

## [Update Statements](https://laravel.com/docs/8.x/queries#update-statements)

$affected = DB::table('users')

->where('id', 1)

->update(['votes' => 1]);

#### [Update Or Insert](https://laravel.com/docs/8.x/queries#update-or-insert)

The updateOrInsert method will attempt to locate a matching database record using the first argument's column and value pairs. If the record exists, it will be updated with the values in the second argument. If the record can not be found, a new record will be inserted with the merged attributes of both arguments:

DB::table('users')

->updateOrInsert(

['email' => 'john@example.com', 'name' => 'John'],

['votes' => '2']

);

### [Increment & Decrement](https://laravel.com/docs/8.x/queries#increment-and-decrement)

these methods accept at least one argument: the column to modify. A second argument may be provided to specify the amount by which the column should be incremented or decremented:

DB::table('users')->increment('votes');

DB::table('users')->increment('votes', 5);

DB::table('users')->decrement('votes');

DB::table('users')->decrement('votes', 5);

## [Delete Statements](https://laravel.com/docs/8.x/queries#delete-statements)

DB::table('users')->delete();

DB::table('users')->where('votes', '>', 100)->delete();

If you wish to truncate an entire table, which will remove all records from the table and reset the auto-incrementing ID to zero, you may use the truncate method:

DB::table('users')->truncate();