Learn with Rabbil

Meet Query Builder

- Allows to interact with your database
- Provides a wide range of methods
- Highly flexible easy to use
- Used to perform complex queries
- Well-documented

Learn with Rabbil

Retrieving All Rows

- Import the DB facade.
- Use the table() method
- Use the get() method to execute

```php
use Illuminate\Support\Facades\DB;
class DemoController extends Controller
{
    function DemoAction(){
        $products = DB::table('products')->get();
        return $products;
    }
}
```

Retrieving Single Row

```php
function DemoAction(){
    $products = DB::table('products')->first();
    return $products;
}
```

```php
function DemoAction(){
    $products = DB::table('products')->find(1);
    return $products;
}
```

Learn with Rabbil

Retrieving List Of Column Values

```php
function DemoAction(){
    $products = DB::table('products')->pluck('price');
    return $products;
}
```

```php
function DemoAction(){
    $products = DB::table('products')->pluck('title', 'id');
    return $products;
}
```

Learn with Rabbil

Aggregates

- The query builder provides a variety of methods for retrieving aggregate values like count, max, min, avg, and sum.
- Call any of these methods after constructing query

```php
function DemoAction(){

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

    $max= DB::table('products')->max('price');
    $avg= DB::table('products')->avg('price');
    $min= DB::table('products')->min('price');
    $sum= DB::table('products')->sum('price');

    return ['count'=>$count,'max'=>$max,'avg'=>$avg,'min'=>$min,'sum'=>$sum];
}
```

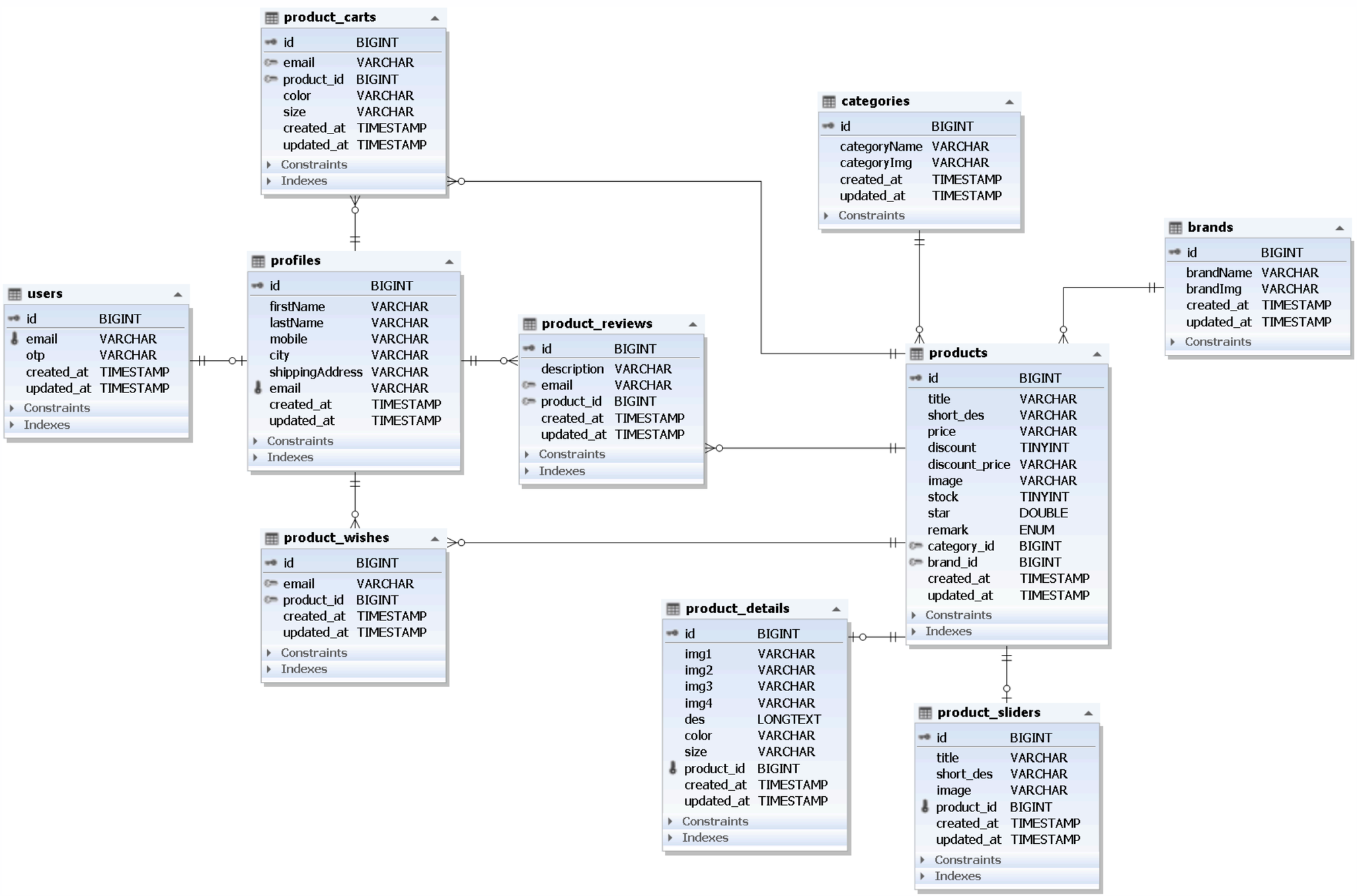
Select Clause

- The select() method allows you to specify the columns

- To return distinct results use the distinct() method

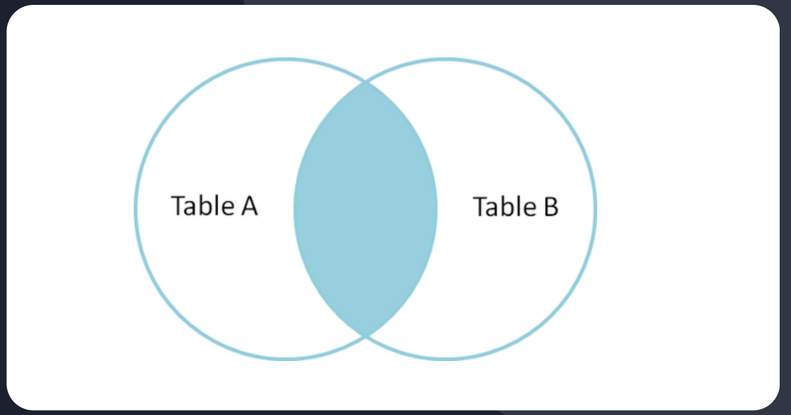
```php
function DemoAction(){
    $products = DB::table('products')->select('title', 'price')->get();
    return $products;
}
```

```php
function DemoAction(){
    $products= DB::table('products')->select('title')->distinct()->get();
    return $products;
}
```


product_carts
id BIGINT
email VARCHAR
product_id BIGINT
color VARCHAR
size VARCHAR
created_at TIMESTAMP
updated_at TIMESTAMP
Constraints
Indexes
categories
id BIGINT
categoryName VARCHAR
categoryImg VARCHAR
created_at TIMESTAMP
updated_at TIMESTAMP
Constraints
brands
id BIGINT
brandName VARCHAR
brandImg VARCHAR
created_at TIMESTAMP
updated_at TIMESTAMP
Constraints
profiles
id BIGINT
firstName VARCHAR
lastName VARCHAR
mobile VARCHAR
city VARCHAR
shippingAddress VARCHAR
email VARCHAR
created_at TIMESTAMP
updated_at TIMESTAMP
Constraints
Indexes
users
id BIGINT
email VARCHAR
otp VARCHAR
created_at TIMESTAMP
updated_at TIMESTAMP
Constraints
Indexes
product_reviews
id BIGINT
description VARCHAR
email VARCHAR
product_id BIGINT
created_at TIMESTAMP
updated_at TIMESTAMP
Constraints
Indexes
products
id BIGINT
title VARCHAR
short_des VARCHAR
price VARCHAR
discount TINYINT
discount_price VARCHAR
image VARCHAR
stock TINYINT
star DOUBLE
remark ENUM
category_id BIGINT
brand_id BIGINT
created_at TIMESTAMP
updated_at TIMESTAMP
Constraints
Indexes
product_wishes
id BIGINT
email VARCHAR
product_id BIGINT
created_at TIMESTAMP
updated_at TIMESTAMP
Constraints
Indexes
product_details
id BIGINT
img1 VARCHAR
img2 VARCHAR
img3 VARCHAR
img4 VARCHAR
des LONGTEXT
color VARCHAR
size VARCHAR
product_id BIGINT
created_at TIMESTAMP
updated_at TIMESTAMP
Constraints
Indexes
product_sliders
id BIGINT
title VARCHAR
short_des VARCHAR
image VARCHAR
product_id BIGINT
created_at TIMESTAMP
updated_at TIMESTAMP
Constraints
Indexes

Inner Join

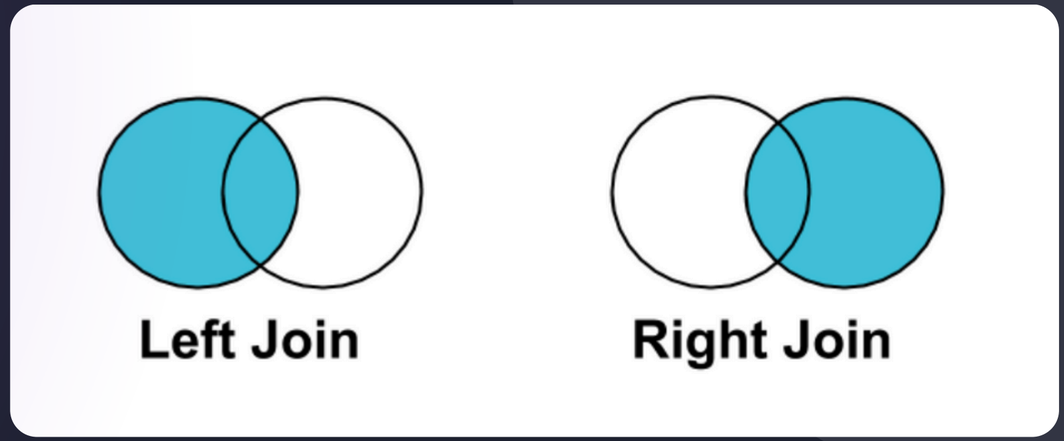
- The name of the table to join
- The column on the current table to join on
- The column on the joined table to join on




```php
function DemoAction(){
    $products= DB::table('products')
            ->join('categories', 'products.category_id', '=', 'categories.id')
            ->join('brands', 'products.brand_id', '=', 'brands.id')
            ->get();
    return $products;
}
```

Left Join Right Join

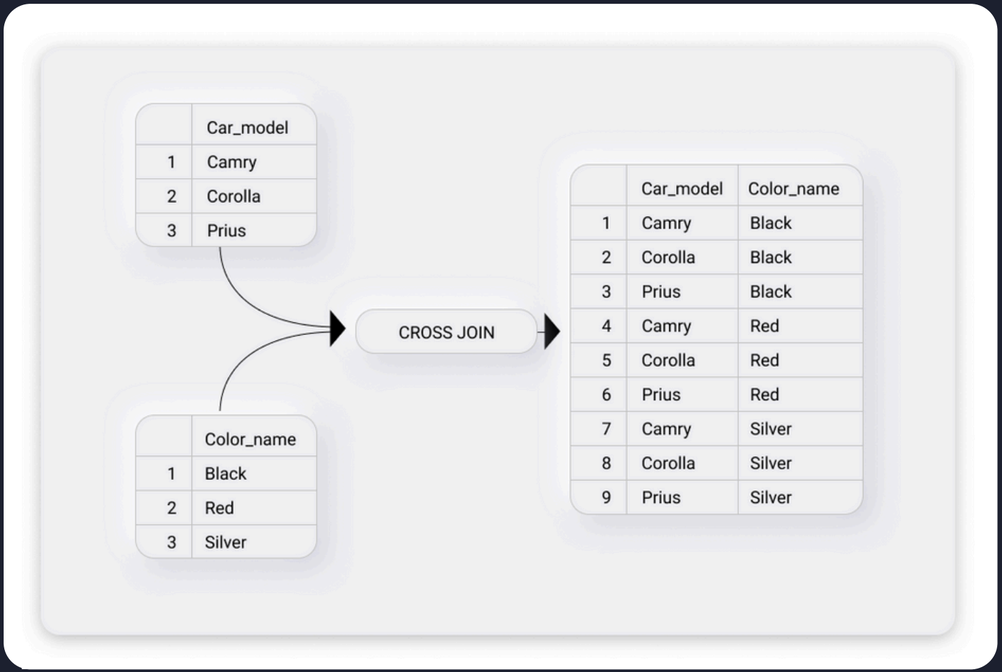
- The name of the table to join
- The column on the current table to join on
- The column on the joined table to join on




```php
function DemoAction(){
    $products= DB::table('products')
            ->leftJoin('categories', 'products.category_id', '=', 'categories.id')
            ->leftJoin('brands', 'products.brand_id', '=', 'brands.id')
            ->get();
    return $products;
}
```

Cross Join

The crossJoin() method returns all possible combinations of rows from the two tables.




```php
function DemoAction(){
    $affected = DB::table('products')
        ->crossJoin('brands')
        ->get();
    return  $affected;
}
```

Advanced Join Clauses

```php
function DemoAction(){
    $products=DB::table('products')
        ->join('categories',function (JoinClause $join){
            $join->on('products.category_id', '=', 'categories.id')
                ->where('products.price', '>', 2000);
        })
        ->join('brands',function (JoinClause $join){
            $join->on('products.brand_id', '=', 'brands.id')
            ->where('brands.brandName', '=', 'Hatil');
        })
        ->get();
    return $products;
}
```

Unions

- The union method takes two arguments: the first argument is the first query, and the second argument is the second query.
- The two queries must select the same columns in the same order.

```php
function DemoAction(){
    $query = DB::table('products')->where('price','>',2000);
    $otherQuery = DB::table('products')->where('category_id','=',3)->union($query)->get();
    return  $otherQuery ;
}
```

Basic Where Clauses

The where() method allows you to filter the results.

- = (equal to)
- != (not equal to)
- < (less than)
- <= (less than or equal to)
- > (greater than)
- >= (greater than or equal to)
- LIKE (contains)
- NOT LIKE (does not contain)
- IN (is in the list)
- NOT IN (is not in the list)

```php
function DemoAction(){
    $products = DB::table('products')->where('price','>',2000)->get();
    return $products;
}
```

```php
function DemoAction(){
    $products = DB::table('products')->where('title','LIKE','%CA%')->get();
    return $products;
}
```

Advance Where Clauses

- The orWhere method to join a clause to the query using the or operator.
- The whereNot and orWhereNot methods may be used to negate a given group of query constraints.
- The whereBetween method verifies that a column's value is between two values .
- The whereNotBetween method verifies that a column's value lies outside of two values.
- The whereBetweenColumns method verifies that a column's value is between the two values of two columns in the same table row.
- The whereNotBetweenColumns method verifies that a column's value lies outside the two values of two columns in the same table row.
- The whereIn method verifies that a given column's value is contained within the given array.
- The whereNotIn method verifies that the given column's value is not contained in the given array.
- The whereNull method verifies that the value of the given column is NULL.
- he whereNotNull method verifies that the column's value is not NULL.
- The whereDate method may be used to compare a column's value against a date.
- The whereMonth method may be used to compare a column's value against a specific month.
- The whereDay method may be used to compare a column's value against a specific day of the month.
- The whereYear method may be used to compare a column's value against a specific year.
- The whereTime method may be used to compare a column's value against a specific time.
- The whereColumn method may be used to verify that two columns are equal.

Advance Where Clauses

```php
function DemoAction(){
    $affected = DB::table('products')
        ->where('price', '>', 8000)
        ->orWhere('title', 'LIKE','%car%')
        ->get();
    return  $affected;
}
```

```php
function DemoAction(){
    $affected = DB::table('products')
        ->where('price', '>', 8000)
        ->whereNot('title', 'LIKE','%car%')
        ->get();
    return  $affected;
}
```

Advance Where Clauses

```php
function DemoAction(){
    $affected = DB::table('products')
        ->whereBetween('price',  [1, 100])
        ->get();
    return  $affected;
}
```

```php
function DemoAction(){
    $affected = DB::table('products')
        ->whereNotBetween('price',  [1, 100])
        ->get();
    return  $affected;
}
```

Advance Where Clauses

```php
function DemoAction(){
    $affected = DB::table('products')
        ->whereIn('price',  [100, 2000, 3000])
        ->get();
    return  $affected;
}
```

```php
function DemoAction(){
    $affected = DB::table('products')
        ->whereNotIn('price',  [100, 2000, 3000])
        ->get();
    return  $affected;
}
```

Advance Where Clauses

```php
function DemoAction(){
    $affected = DB::table('products')
        ->whereNull('price')
        ->get();
    return  $affected;
}
```

```php
function DemoAction(){
    $affected = DB::table('products')
        ->whereNotNull('price')
        ->get();
    return  $affected;
}
```

Advance Where Clauses

```php
function DemoAction(){
    $affected = DB::table('products')
        ->whereDate('updated_at', '2023-05-16')
        ->get();
    return  $affected;
}
```

```php
function DemoAction(){
    $affected = DB::table('products')
        ->whereMonth('updated_at', '05')
        ->get();
    return  $affected;
}
```

Advance Where Clauses

```php
function DemoAction(){
    $affected = DB::table('products')
        ->whereDay('updated_at', '05')
        ->get();
    return  $affected;
}
```

```php
function DemoAction(){
    $affected = DB::table('products')
        ->whereYear('updated_at', '2023')
        ->get();
    return  $affected;
}
```

Advance Where Clauses

```php
function DemoAction(){
    $affected = DB::table('products')
        ->whereTime('updated_at', '01:25:04')
        ->get();
    return  $affected;
}
```

```php
function DemoAction(){
    $affected = DB::table('products')
        ->whereColumn('updated_at', '>', 'created_at')
        ->get();
    return  $affected;
}
```

Ordering, Grouping, Limit

- The orderBy method allows you to sort the results of the query by a given column
- The latest and oldest methods allow you to easily order results by date
- The inRandomOrder method may be used to sort the query results randomly
- The groupBy and having methods may be used to group the query results
- 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

Ordering

```php
function DemoAction(){
    $affected = DB::table('brands')
        ->orderBy('brandName', 'desc')
        ->get();
    return  $affected;
}
```

```php
function DemoAction(){
    $affected = DB::table('brands')
    ->inRandomOrder()
    ->first();
    return  $affected;
}
```

Learn with Rabbil

Latest & Oldest

```php
function DemoAction(){
    $affected = DB::table('brands')
    ->latest()
    ->first();
    return  $affected;
}
```

```php
function DemoAction(){
    $affected = DB::table('brands')
    ->oldest()
    ->first();
    return  $affected;
}
```

groupBy and having

```php
function DemoAction(){
    $affected = DB::table('products')
        ->groupBy('price')
        ->get();
    return  $affected;
}
```

```php
function DemoAction(){
    $affected = DB::table('products')
        ->groupBy('price')
        ->having('price', '>', 5000)
        ->get();
    return  $affected;
}
```

Skip & Take

```php
function DemoAction(){
    $affected = DB::table('products')
        ->skip(10)
        ->take(5)
        ->get();
    return  $affected;
}
```

Insert Statements

- Insert method used to insert records into the database table.
- The insert method accepts an array of column names and values.

```php
function DemoAction(){
    $affected = DB::table('brands')
    ->insert(
        ['brandName' => 'New brand','brandImg' => 'New brand img']
    );
    return  $affected;
}
```

Update Statements

- Update existing records using the update method

```php
function DemoAction(){
    $affected = DB::table('products')
    ->where('id', 1)
    ->update(['price' => 1000]);
}
```

Update or Insert Statements

- updateOrInsert method may be used. to update an existing record in the database or create it if no matching record exists.

```php
function DemoAction(){
    $affected = DB::table('brands')
    ->updateOrInsert(
        ['brandName'=>'Hatil'],
        ['brandName' => 'New Hatil','brandImg' => 'Hatil']
    );
    return  $affected;
}
```

Increment & Decrement

- The query builder also provides convenient methods for incrementing or decrementing the value of a given column.

```php
function DemoAction(){
    $result=DB::table('products')
        ->where('id','=',1)
        ->increment('price');
        // ->increment('price',10);
        // ->decrement('price');
        // ->decrement('price',10);

    return  $result ;
}
```

Delete Statements

The query builder's delete method may be used to delete records from the table to truncate an entire table use the truncate method

```php
function DemoAction(Request $request){
    $deleted = DB::table('products')
    ->where('id', '=', $request->id)
    ->delete();
    return  $deleted ;
}
```

```php
function DemoAction(Request $request){
    $deleted = DB::table('users')->truncate();
    return  $deleted ;
}
```

Learn with Rabbil

Paginate

Display simple "Next" and "Previous" links in your application's UI, use the simplePaginate method to perform a single, efficient query

```php
function DemoAction(){
    $affected = DB::table('products')
        ->simplePaginate(2);
    return  $affected;
}
```

Paginate

The paginate method counts the total number of records matched by the query before retrieving the records from the database.

```php
function DemoAction(){
    $affected = DB::table('products')
        ->paginate(2);
    return  $affected;
}
```

```php
function DemoAction(){
    $affected = DB::table('products')
        ->paginate(
            $perPage = 2, $columns = ['*'], $pageName = 'products'
        );
    return  $affected;
}
```