HOW TO USE MONGODB $UNWIND

MongoDB $unwind is a useful tool in performing aggregation in MongoDB. Let's take a look at $unwind and how to use it.

(This article is part of our MongoDB Guide. Use the right-hand menu to navigate.)

What is MongoDB $unwind?

The MongoDB $unwind operator is used to deconstruct an array field in a document and create separate output documents for each item in the array.

The only difference between the input document and output documents is that, in output documents, the value of the array field is replaced by a single item from the input document array. You will see this as I walk you through this example.

MongoDB $unwind transforms complex documents into simpler documents, which increase readability and understanding. This also allows us to perform additional operations, like grouping and sorting on the resulting output.

The basic syntax of $unwind operator

```
{ $unwind: "$<field path/ array path>" }
```

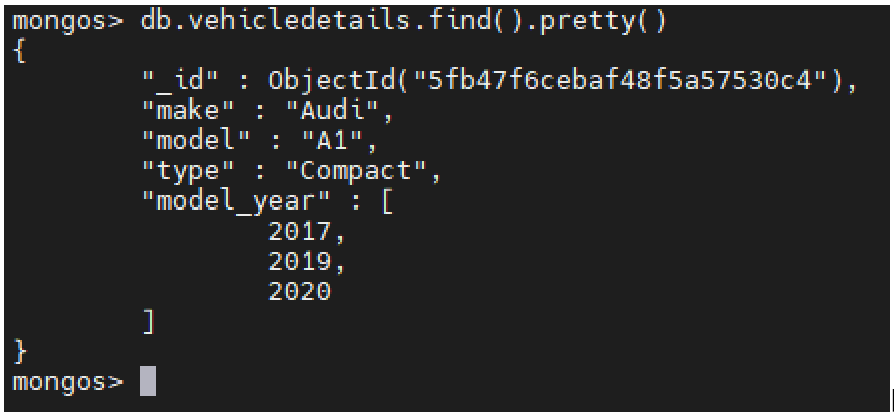
```
{$unwind: { path : "$<field path/ array path>", <optional arguments>}}
```

Here, you should prefix the path with the dollar sign "$" for a successful $unwind operation.

Next, we will demonstrate a simple $unwind operation using our "vehicledetails" collection. First, let's have a look at our collection.

```
db.vehicledetails.find().pretty()
```

Result:

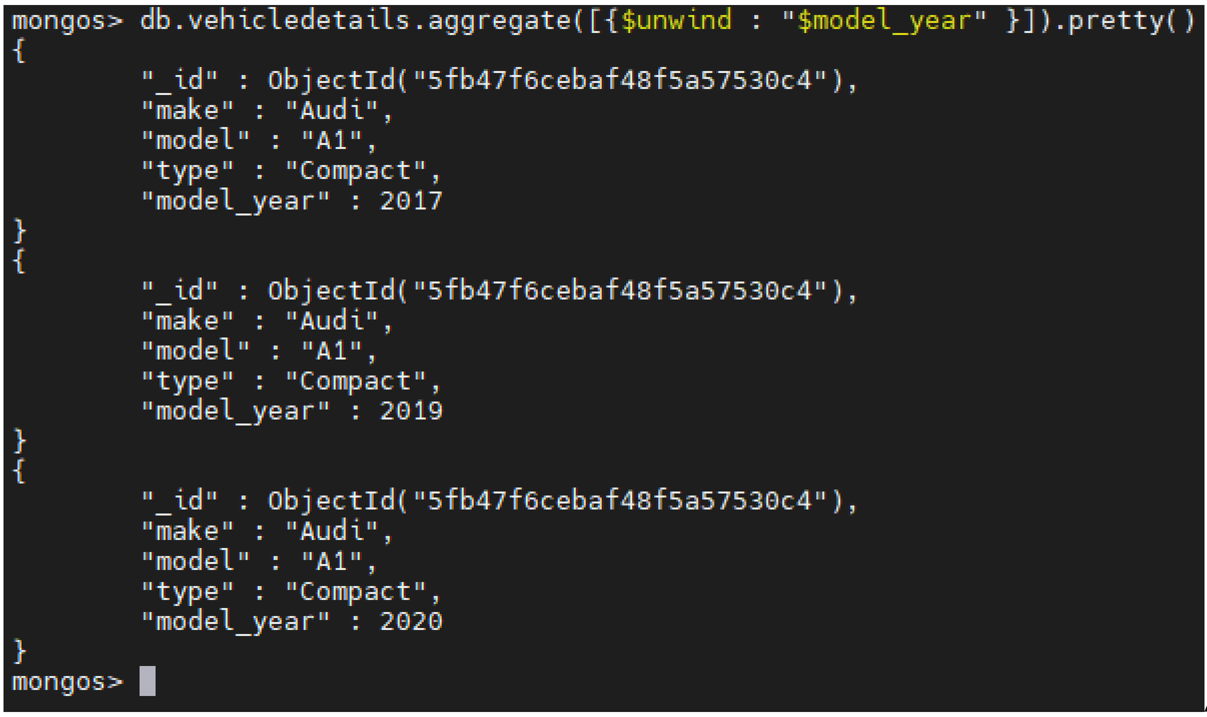


```
mongos> db.vehicledetails.find().pretty()
{
        "_id" : ObjectId("5fb47f6cebaf48f5a57530c4"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : [
                2017,
                2019,
                2020
        ]
}
mongos>
```

Next, we use the $unwind operator to deconstruct the "model_year" array field and to create separate documents for each year.

```
db.vehicledetails.aggregate().pretty()
```

Result:



```
mongos> db.vehicledetails.aggregate([{$unwind : "$model_year" }]).pretty()
{
        "_id" : ObjectId("5fb47f6cebaf48f5a57530c4"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : 2017
}
{
        "_id" : ObjectId("5fb47f6cebaf48f5a57530c4"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : 2019
}
{
        "_id" : ObjectId("5fb47f6cebaf48f5a57530c4"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : 2020
}
mongos>
```

As shown in the above output, a new document is created for each item in the "model_year" array.

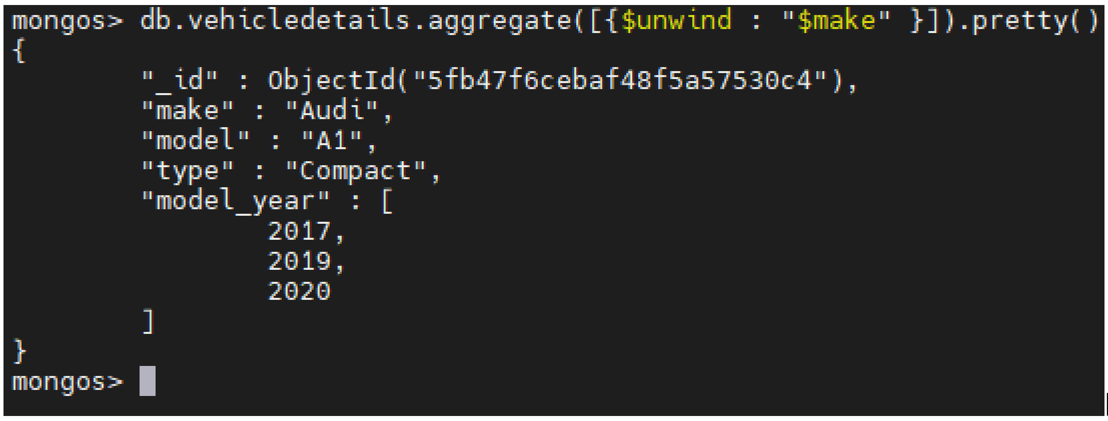
How $unwind works with non-array field paths

Before MongoDB 3.2, if a non-array field is defined as the path in the $unwind operator, it would have resulted in an error. From MongoDB 3.2 onwards, any non-array field which does not resolve to a **missing**, **null**, or **empty array** will be treated as a single element array.

This next example shows how the $unwind operator handles a single element array.

```
db.vehicledetails.aggregate().pretty()
```

Result:



```
mongos> db.vehicledetails.aggregate([{$unwind : "$make" }]).pretty()
{
        "_id" : ObjectId("5fb47f6cebaf48f5a57530c4"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : [
                2017,
                2019,
                2020
        ]
}
mongos>
```

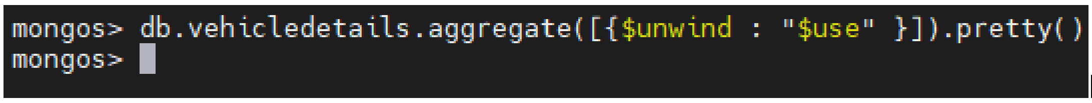
Here, the $unwind operation successfully creates a single output document as it considers the "make" field as a single item array.

Missing field as the path

The $unwind operator will not generate any output if the specified path is a missing/unavailable field. There will be no error as the $unwind will ignore the input document.

```
db.vehicledetails.aggregate().pretty()
```

Result:



```
mongos> db.vehicledetails.aggregate([{$unwind : "$use" }]).pretty()
mongos>
```

In the above operation, the provided path for the $unwind operator is the "use" field. However, the input document is ignored, as there is no "use" field in the input document.

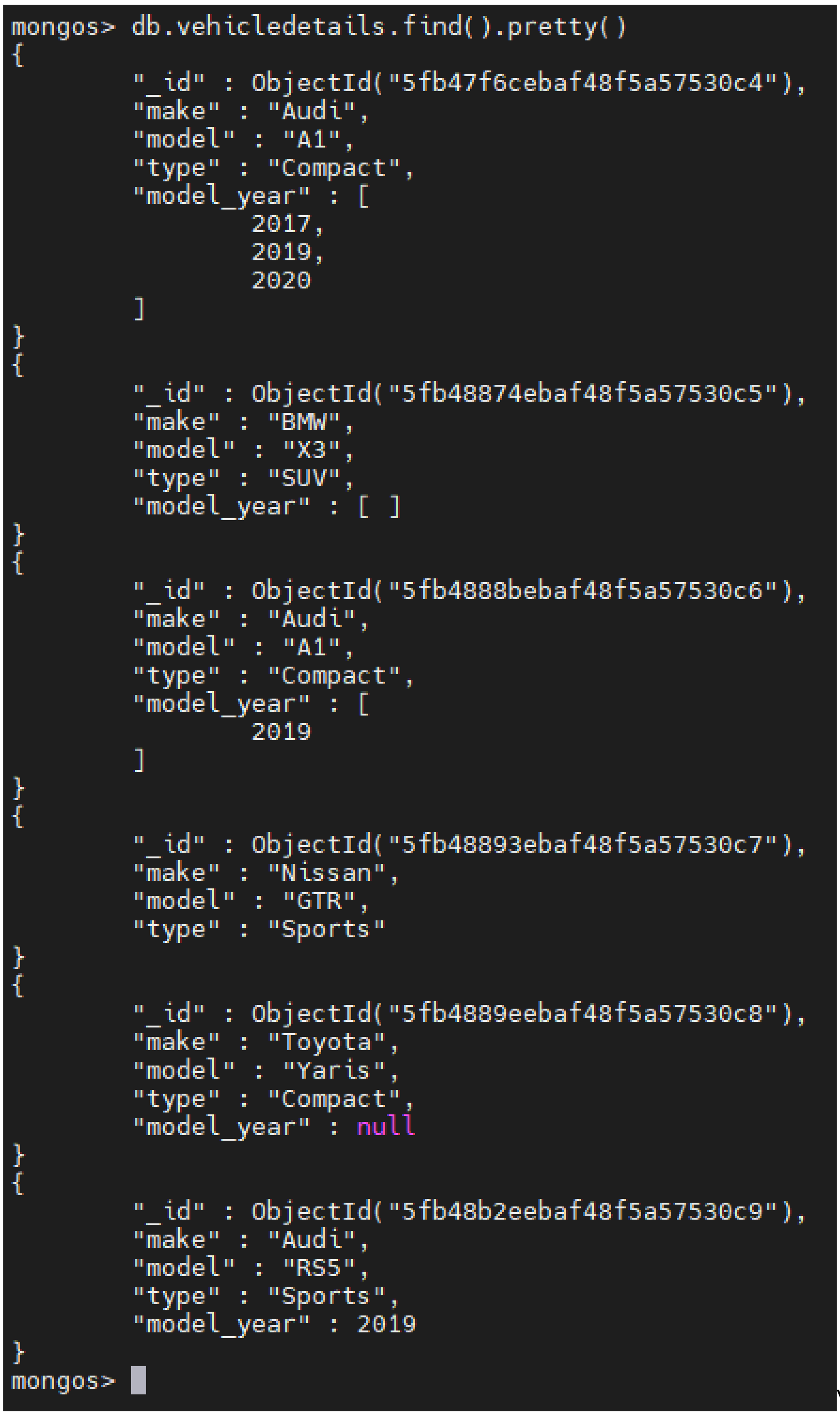
$unwind operator options

The $unwind operator can be used with two optional arguments, which are **includeArrayIndex** and **preserveNullAndEmptyArrays**. This section will demonstrate how each option affects the result of an $unwind operation.

Again, we'll use the "vehicledetails" collection with additional documents to demonstrate the functionality of the optional arguments.

```
db.vehicledetails.find().pretty()
```

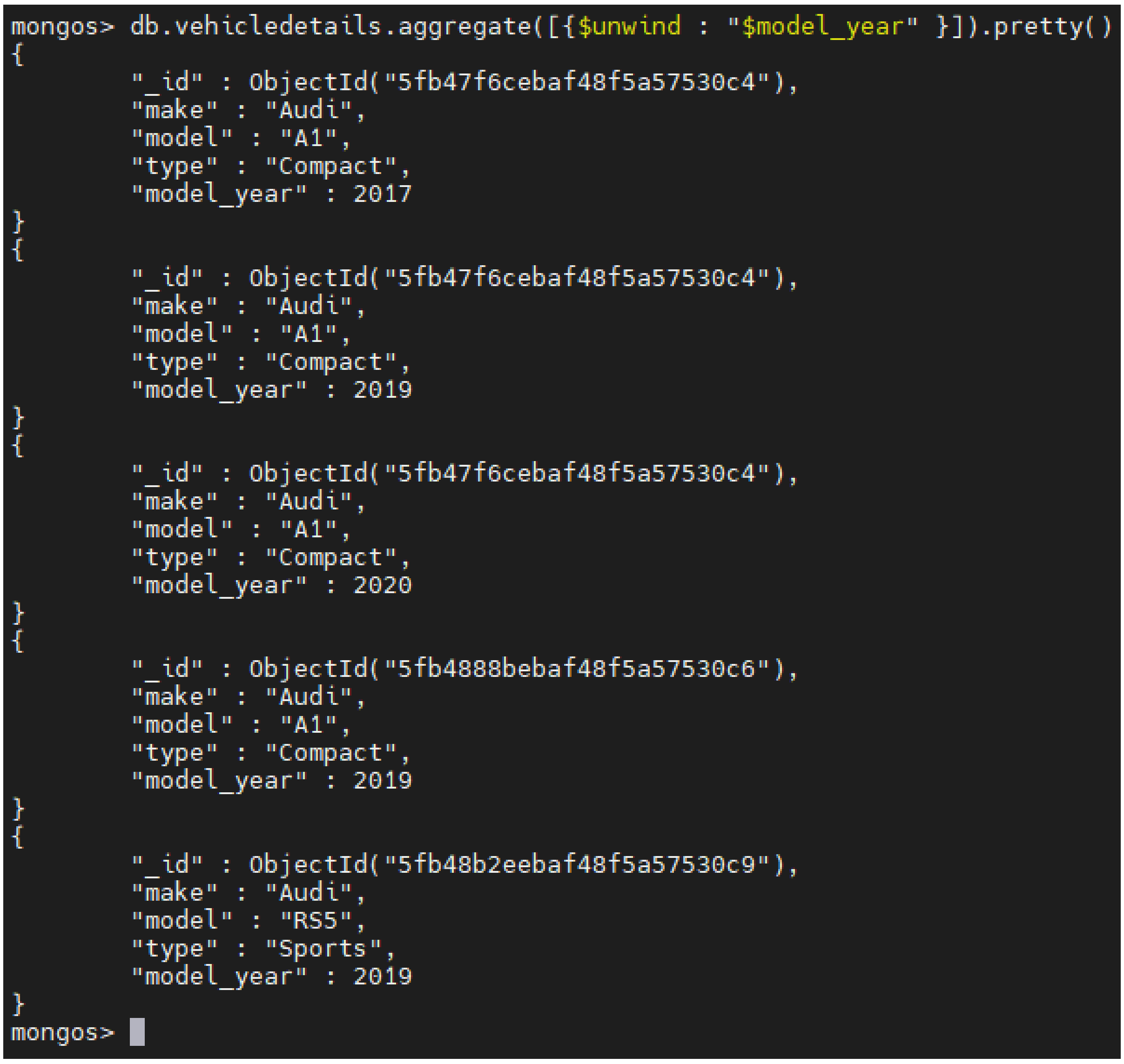
Result:


```
mongos> db.vehicledetails.find().pretty()
{
        "_id" : ObjectId("5fb47f6cebaf48f5a57530c4"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : [
                2017,
                2019,
                2020
        ]
}
{
        "_id" : ObjectId("5fb48874ebaf48f5a57530c5"),
        "make" : "BMW",
        "model" : "X3",
        "type" : "SUV",
        "model_year" : [ ]
}
{
        "_id" : ObjectId("5fb4888bebaf48f5a57530c6"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : [
                2019
        ]
}
{
        "_id" : ObjectId("5fb48893ebaf48f5a57530c7"),
        "make" : "Nissan",
        "model" : "GTR",
        "type" : "Sports"
}
{
        "_id" : ObjectId("5fb4889eebaf48f5a57530c8"),
        "make" : "Toyota",
        "model" : "Yaris",
        "type" : "Compact",
        "model_year" : null
}
{
        "_id" : ObjectId("5fb48b2eebaf48f5a57530c9"),
        "make" : "Audi",
        "model" : "RS5",
        "type" : "Sports",
        "model_year" : 2019
}
mongos>
```
When you perform $unwind operation on the above data set with the "model_year" field, the output will consist of five results.

Let's see this in this example:

```
db.vehicledetails.aggregate().pretty()
```

Result:



```
mongos> db.vehicledetails.aggregate([{$unwind : "$model_year" }]).pretty()
{
        "_id" : ObjectId("5fb47f6cebaf48f5a57530c4"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : 2017
}
{
        "_id" : ObjectId("5fb47f6cebaf48f5a57530c4"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : 2019
}
{
        "_id" : ObjectId("5fb47f6cebaf48f5a57530c4"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : 2020
}
{
        "_id" : ObjectId("5fb4888bebaf48f5a57530c6"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : 2019
}
{
        "_id" : ObjectId("5fb48b2eebaf48f5a57530c9"),
        "make" : "Audi",
        "model" : "RS5",
        "type" : "Sports",
        "model_year" : 2019
}
mongos>
```

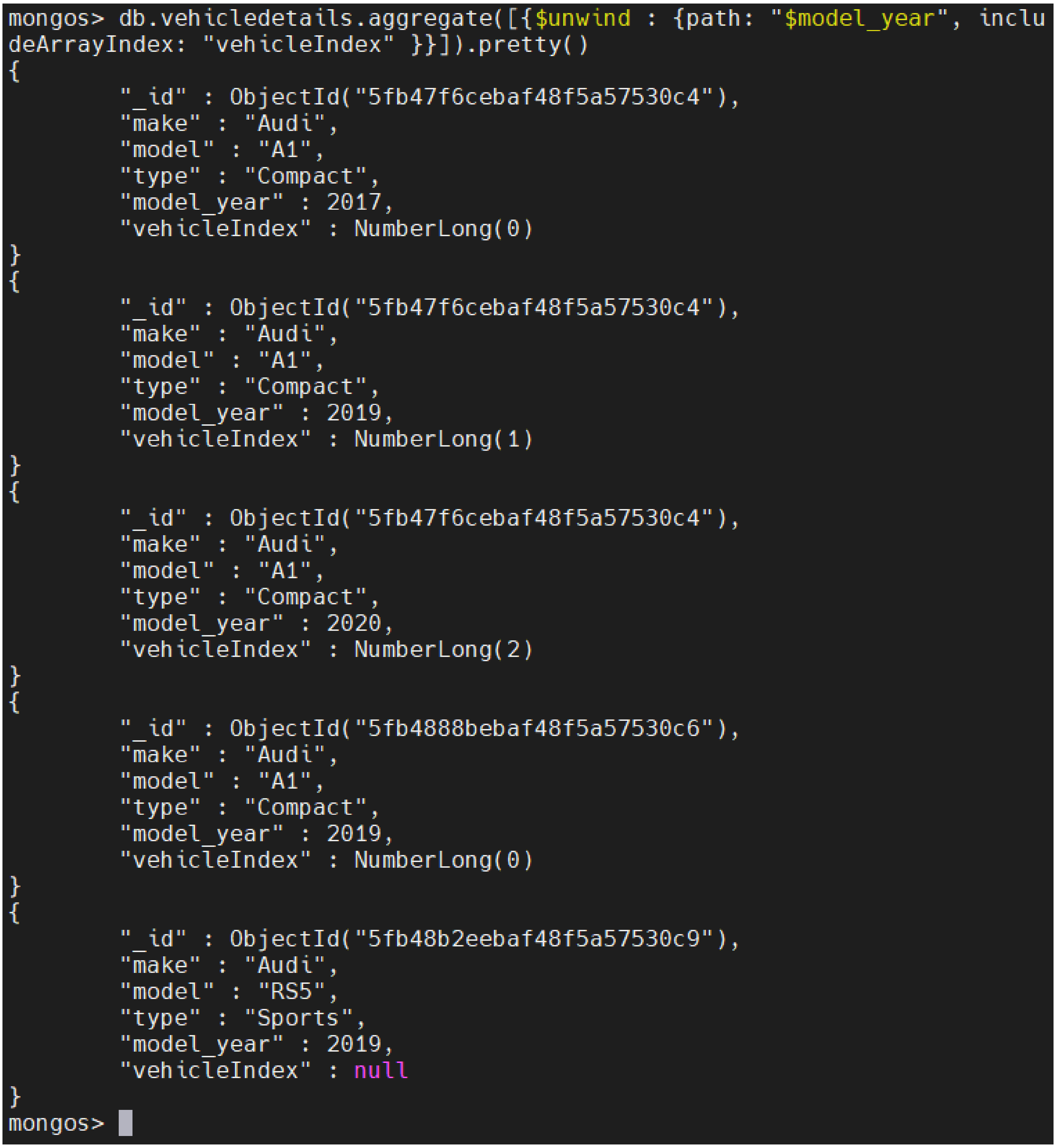
includeArrayIndex option

The **includeArrayIndex** option is used to obtain the array index with the $unwind operation output. We can define a field in the "includeArrayIndex" syntax to get the array index of each output document. If the array field is null, this will result in a null value in the user-defined field, and if an empty array is present, it will be ignored as there are no items in that array.

In the below example, the "model_year" field is used to carry out the $unwind operation, and the user-defined "vehicleIndex" field is used to capture the array index.

```
db.vehicledetails.aggregate().pretty()
```

Result:



```
mongos> db.vehicledetails.aggregate([{$unwind : {path: "$model_year", inclu
deArrayIndex: "vehicleIndex" }}]).pretty()
{
        "_id" : ObjectId("5fb47f6cebaf48f5a57530c4"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : 2017,
        "vehicleIndex" : NumberLong(0)
}
{
        "_id" : ObjectId("5fb47f6cebaf48f5a57530c4"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : 2019,
        "vehicleIndex" : NumberLong(1)
}
{
        "_id" : ObjectId("5fb47f6cebaf48f5a57530c4"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : 2020,
        "vehicleIndex" : NumberLong(2)
}
{
        "_id" : ObjectId("5fb4888bebaf48f5a57530c6"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : 2019,
        "vehicleIndex" : NumberLong(0)
}
{
        "_id" : ObjectId("5fb48b2eebaf48f5a57530c9"),
        "make" : "Audi",
        "model" : "RS5",
        "type" : "Sports",
        "model_year" : 2019,
        "vehicleIndex" : null
}
mongos>
```

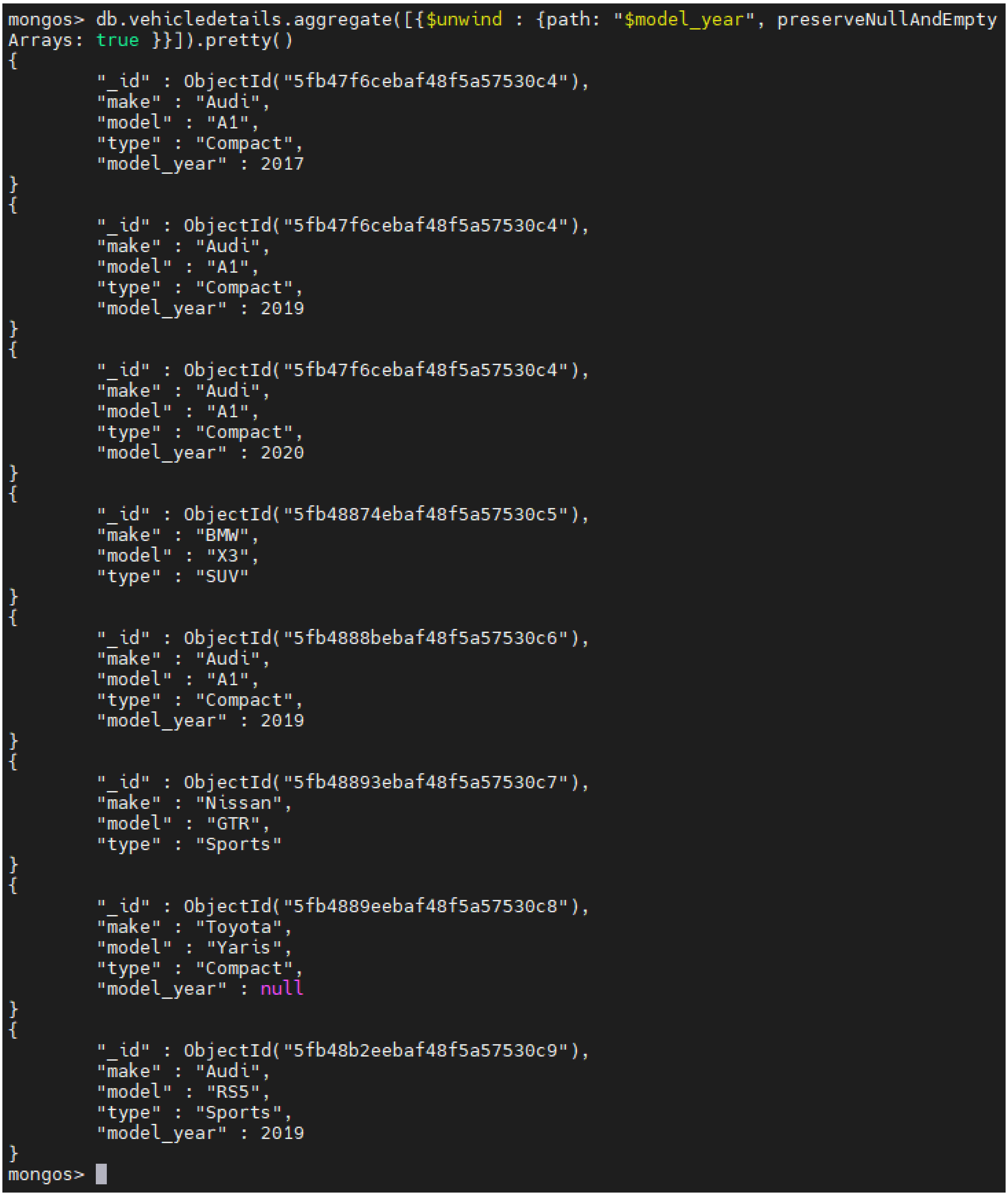
preserveNullAndEmptyArrays option

Defining the "preserveNullAndEmptyArrays" option in a $unwind operation allows the user to include documents where the array field is missing, null, or an empty array. The "preserveNullAndEmptyArrays" option only takes Boolean arguments like true or false.

Let's use the documents within the "vehicledetails" collection to demonstrate "preserveNullAndEmptyArrays" option. This will result in an output where all missing, null, or empty arrays in the "model_year" field are captured.

```
db.vehicledetails.aggregate().pretty()
```

Result:



```
mongos> db.vehicledetails.aggregate([{$unwind : {path: "$model_year", preserveNullAndEmpty
Arrays: true }}]).pretty()
{
        "_id" : ObjectId("5fb47f6cebaf48f5a57530c4"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : 2017
}
{
        "_id" : ObjectId("5fb47f6cebaf48f5a57530c4"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : 2019
}
{
        "_id" : ObjectId("5fb47f6cebaf48f5a57530c4"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : 2020
}
{
        "_id" : ObjectId("5fb48874ebaf48f5a57530c5"),
        "make" : "BMW",
        "model" : "X3",
        "type" : "SUV"
}
{
        "_id" : ObjectId("5fb4888bebaf48f5a57530c6"),
        "make" : "Audi",
        "model" : "A1",
        "type" : "Compact",
        "model_year" : 2019
}
{
        "_id" : ObjectId("5fb48893ebaf48f5a57530c7"),
        "make" : "Nissan",
        "model" : "GTR",
        "type" : "Sports"
}
{
        "_id" : ObjectId("5fb4889eebaf48f5a57530c8"),
        "make" : "Toyota",
        "model" : "Yaris",
        "type" : "Compact",
        "model_year" : null
}
{
        "_id" : ObjectId("5fb48b2eebaf48f5a57530c9"),
        "make" : "Audi",
        "model" : "RS5",
        "type" : "Sports",
        "model_year" : 2019
}
mongos>
```

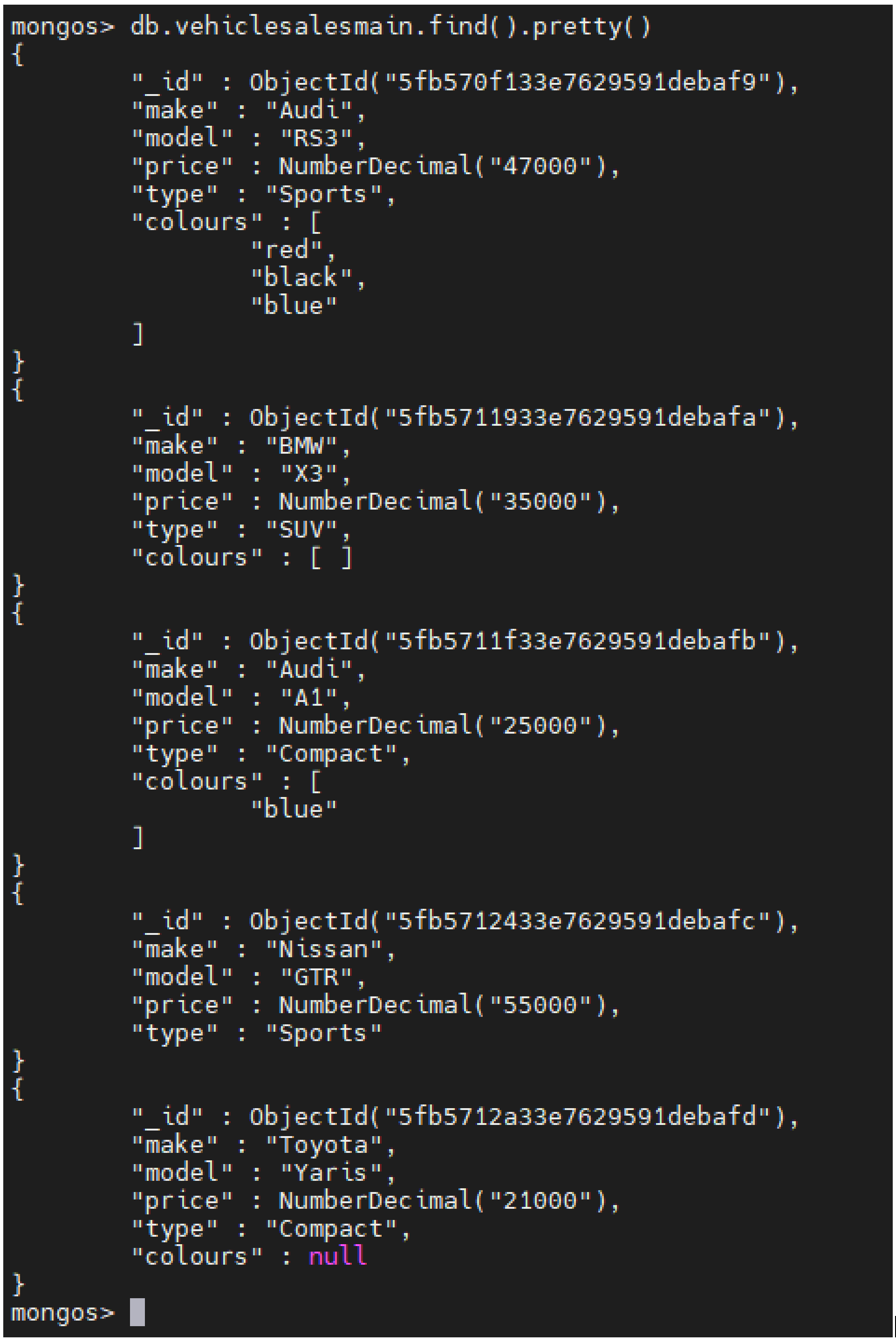
Grouping data by Unwound values

In MongoDB, you can use the resulting dataset of a $unwind operation—unwound values—to further transform the data set.

The “vehiclesalesmain” collection is used to demonstrate how to combine $unwind operator with other functions.

```
db.vehiclesalesmain.find().pretty()
```

Result:

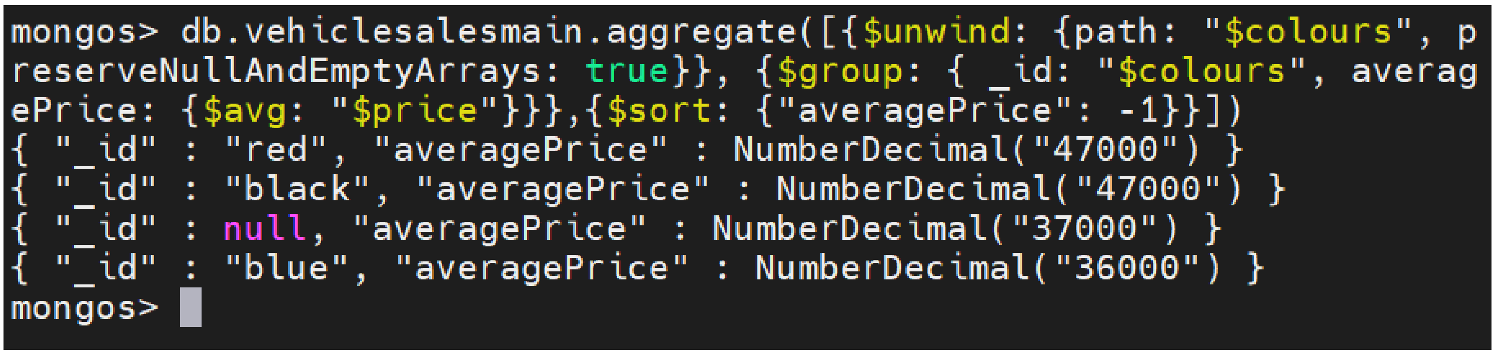


```
mongos> db.vehiclesalesmain.find().pretty()
{
        "_id" : ObjectId("5fb570f133e7629591debaf9"),
        "make" : "Audi",
        "model" : "RS3",
        "price" : NumberDecimal("47000"),
        "type" : "Sports",
        "colours" : [
                "red",
                "black",
                "blue"
        ]
}
{
        "_id" : ObjectId("5fb5711933e7629591debafa"),
        "make" : "BMW",
        "model" : "X3",
        "price" : NumberDecimal("35000"),
        "type" : "SUV",
        "colours" : [ ]
}
{
        "_id" : ObjectId("5fb5711f33e7629591debafb"),
        "make" : "Audi",
        "model" : "A1",
        "price" : NumberDecimal("25000"),
        "type" : "Compact",
        "colours" : [
                "blue"
        ]
}
{
        "_id" : ObjectId("5fb5712433e7629591debafc"),
        "make" : "Nissan",
        "model" : "GTR",
        "price" : NumberDecimal("55000"),
        "type" : "Sports"
}
{
        "_id" : ObjectId("5fb5712a33e7629591debafd"),
        "make" : "Toyota",
        "model" : "Yaris",
        "price" : NumberDecimal("21000"),
        "type" : "Compact",
        "colours" : null
}
mongos> 
```

In the next section, the "colours" array is used to perform the $unwind operation. We also use MongoDB group and sort methods to get the average price according to the vehicle colours.

```
db.vehiclesalesmain.aggregate()
```

Result:



```
mongos> db.vehiclesalesmain.aggregate([{$unwind: {path: "$colours", preserveNullAndEmptyArrays: true}}, {$group: { _id: "$colours", averagePrice: {$avg: "$price"}}},{$sort: {"averagePrice": -1}}])
{ "_id" : "red", "averagePrice" : NumberDecimal("47000") }
{ "_id" : "black", "averagePrice" : NumberDecimal("47000") }
{ "_id" : null, "averagePrice" : NumberDecimal("37000") }
{ "_id" : "blue", "averagePrice" : NumberDecimal("36000") }
mongos>
```

Let’s simplify the above formula in several steps to get a better understanding of the process. You can see the complete formatted syntax below.

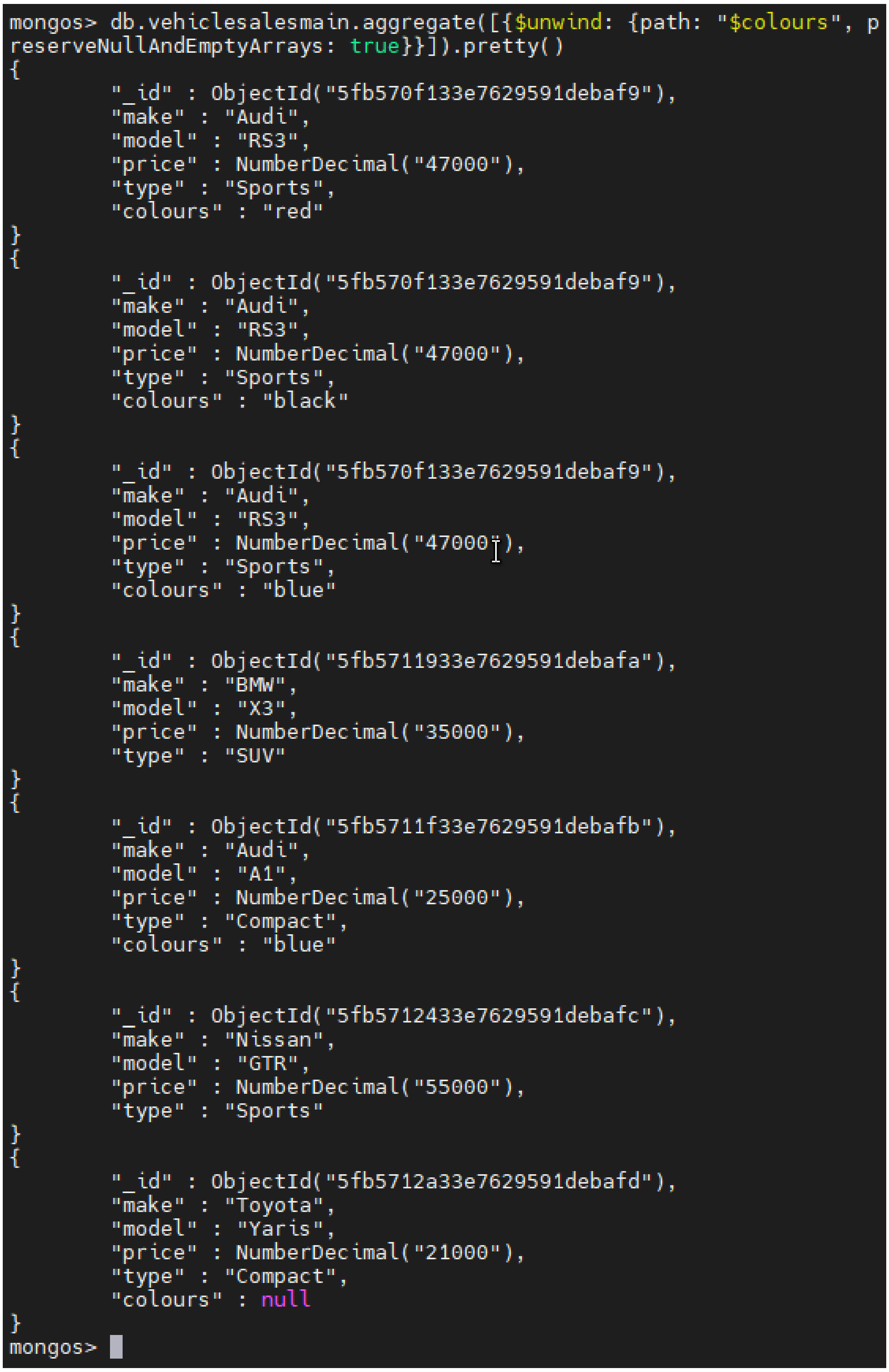
```
db.vehiclesalesmain.aggregate()
```

Step 1

In the first step, the documents in the “vehiclesalesmain" collection are unwound using the "colours" array field. Then the preserveNullAndEmptyArrays is set to true to capture any missing, null, or empty arrays in the resulting output.

```
db.vehiclesalesmain.aggregate().pretty()
```

Result:



```
mongos> db.vehiclesalesmain.aggregate([{$unwind: {path: "$colours", p
reserveNullAndEmptyArrays: true}}]).pretty()
{
        "_id" : ObjectId("5fb570f133e7629591debaf9"),
        "make" : "Audi",
        "model" : "RS3",
        "price" : NumberDecimal("47000"),
        "type" : "Sports",
        "colours" : "red"
}
{
        "_id" : ObjectId("5fb570f133e7629591debaf9"),
        "make" : "Audi",
        "model" : "RS3",
        "price" : NumberDecimal("47000"),
        "type" : "Sports",
        "colours" : "black"
}
{
        "_id" : ObjectId("5fb570f133e7629591debaf9"),
        "make" : "Audi",
        "model" : "RS3",
        "price" : NumberDecimal("47000"),
        "type" : "Sports",
        "colours" : "blue"
}
{
        "_id" : ObjectId("5fb5711933e7629591debafa"),
        "make" : "BMW",
        "model" : "X3",
        "price" : NumberDecimal("35000"),
        "type" : "SUV"
}
{
        "_id" : ObjectId("5fb5711f33e7629591debafb"),
        "make" : "Audi",
        "model" : "A1",
        "price" : NumberDecimal("25000"),
        "type" : "Compact",
        "colours" : "blue"
}
{
        "_id" : ObjectId("5fb5712433e7629591debafc"),
        "make" : "Nissan",
        "model" : "GTR",
        "price" : NumberDecimal("55000"),
        "type" : "Sports"
}
{
        "_id" : ObjectId("5fb5712a33e7629591debafd"),
        "make" : "Toyota",
        "model" : "Yaris",
        "price" : NumberDecimal("21000"),
        "type" : "Compact",
        "colours" : null
}
mongos>
```

Step 2

In the second step, documents resulting from the $unwind operation are grouped by the MongoDB group method using the colour, and the average price for each colour is calculated.

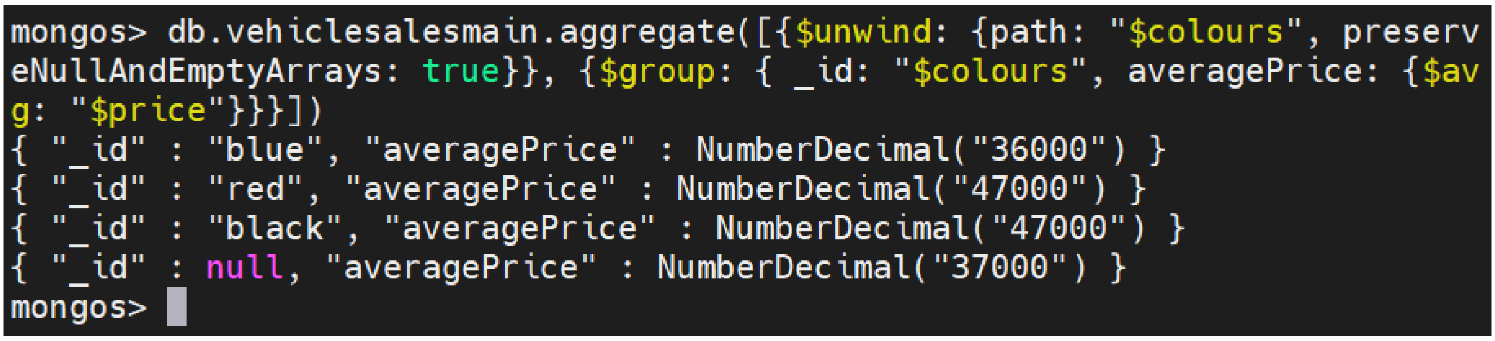
Grouping syntax:

```
{$group: { _id: "$colours", averagePrice: {$avg: "$price"}}}
```

Complete syntax:

```
db.vehiclesalesmain.aggregate()
```

Result:



```
mongos> db.vehiclesalesmain.aggregate([{$unwind: {path: "$colours", preserv
eNullAndEmptyArrays: true}}, {$group: { _id: "$colours", averagePrice: {$av
g: "$price"}}}])
{ "_id" : "blue", "averagePrice" : NumberDecimal("36000") }
{ "_id" : "red", "averagePrice" : NumberDecimal("47000") }
{ "_id" : "black", "averagePrice" : NumberDecimal("47000") }
{ "_id" : null, "averagePrice" : NumberDecimal("37000") }
mongos>
```

Step 3

Finally, the grouped data set is sorted in descending order using the MongoDB sort method.

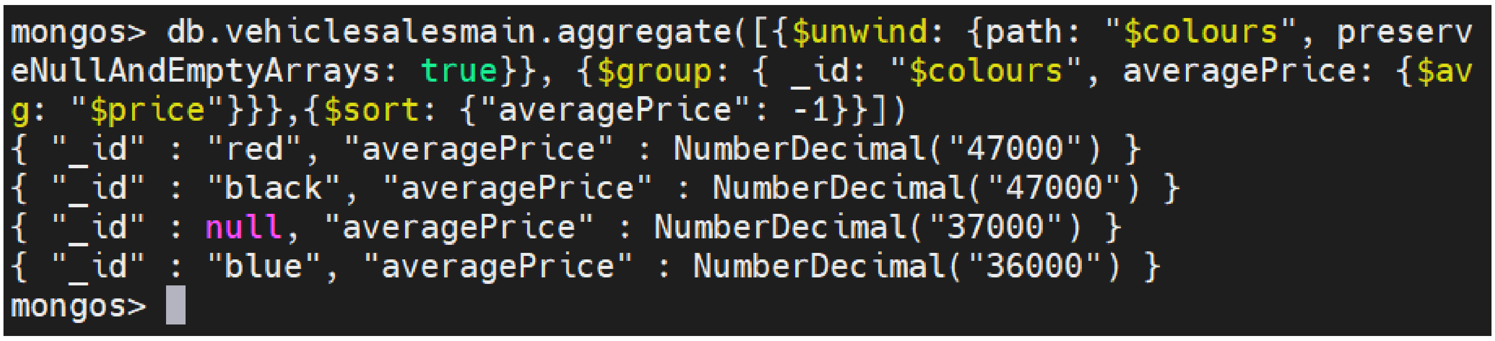
Sorting syntax:

```
{$sort: {"averagePrice": -1}}])
```

Complete syntax:

```
db.vehiclesalesmain.aggregate()
```

Result:



```
mongos> db.vehiclesalesmain.aggregate([{$unwind: {path: "$colours", preserv
eNullAndEmptyArrays: true}}, {$group: { _id: "$colours", averagePrice: {$av
g: "$price"}}},{$sort: {"averagePrice": -1}}])
{ "_id" : "red", "averagePrice" : NumberDecimal("47000") }
{ "_id" : "black", "averagePrice" : NumberDecimal("47000") }
{ "_id" : null, "averagePrice" : NumberDecimal("37000") }
{ "_id" : "blue", "averagePrice" : NumberDecimal("36000") }
mongos>
```

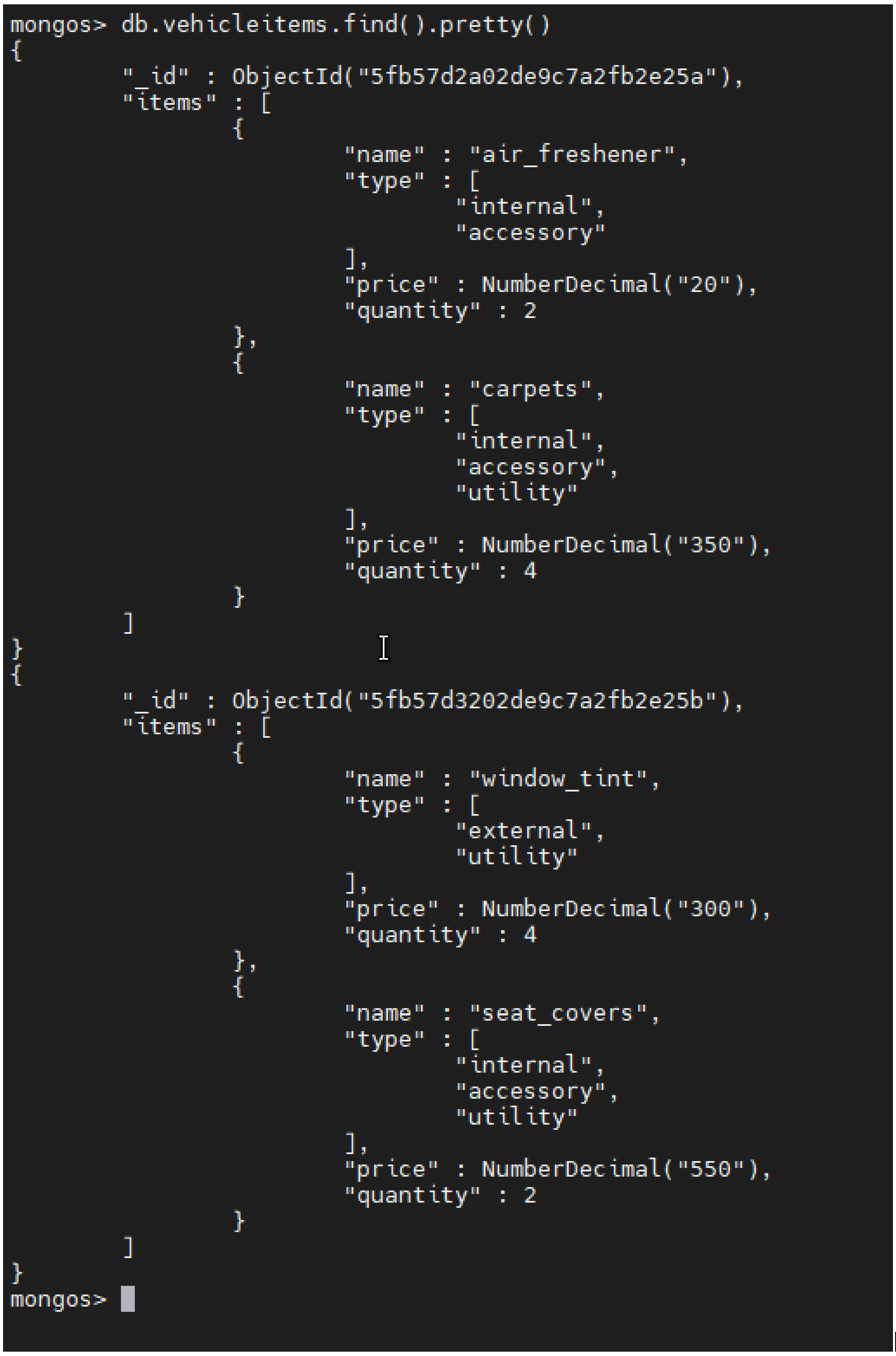
Using $unwind on embedded arrays

When you perform the $unwind operation on an embedded array, it will function the same way as on a normal array field. The $unwind operation will deconstruct the items in each of the embedded arrays.

Using the “vehicleitems" collection, we will demonstrate how the $unwind operator functions on an embedded array. Let's have a look at the "vehicleitems" collection.

```
db.vehicleitems.find().pretty()
```

Result:

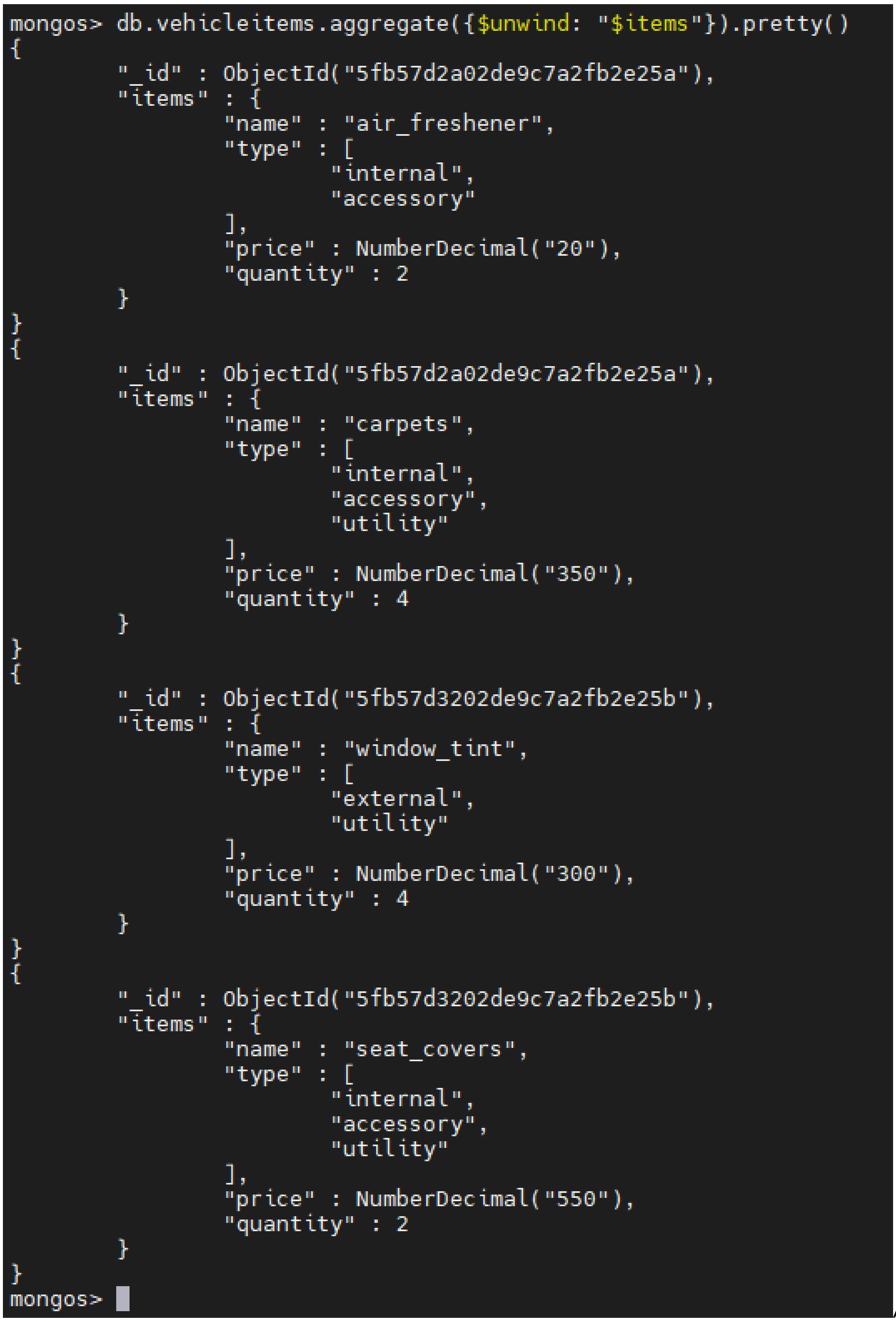


```
mongos> db.vehicleitems.find().pretty()
{
        "_id" : ObjectId("5fb57d2a02de9c7a2fb2e25a"),
        "items" : [
                {
                        "name" : "air_freshener",
                        "type" : [
                                "internal",
                                "accessory"
                        ],
                        "price" : NumberDecimal("20"),
                        "quantity" : 2
                },
                {
                        "name" : "carpets",
                        "type" : [
                                "internal",
                                "accessory",
                                "utility"
                        ],
                        "price" : NumberDecimal("350"),
                        "quantity" : 4
                }
        ]
}
{
        "_id" : ObjectId("5fb57d3202de9c7a2fb2e25b"),
        "items" : [
                {
                        "name" : "window_tint",
                        "type" : [
                                "external",
                                "utility"
                        ],
                        "price" : NumberDecimal("300"),
                        "quantity" : 4
                },
                {
                        "name" : "seat_covers",
                        "type" : [
                                "internal",
                                "accessory",
                                "utility"
                        ],
                        "price" : NumberDecimal("550"),
                        "quantity" : 2
                }
        ]
}
mongos>
```

In the next section, the $unwind operation is performed on the above documents using the “items” embedded arrays.

```
db.vehicleitems.aggregate({$unwind: "$items"}).pretty()
```

Result:



```
mongos> db.vehicleitems.aggregate({$unwind: "$items"}).pretty()
{
        "_id" : ObjectId("5fb57d2a02de9c7a2fb2e25a"),
        "items" : {
                "name" : "air_freshener",
                "type" : [
                        "internal",
                        "accessory"
                ],
                "price" : NumberDecimal("20"),
                "quantity" : 2
        }
}
{
        "_id" : ObjectId("5fb57d2a02de9c7a2fb2e25a"),
        "items" : {
                "name" : "carpets",
                "type" : [
                        "internal",
                        "accessory",
                        "utility"
                ],
                "price" : NumberDecimal("350"),
                "quantity" : 4
        }
}
{
        "_id" : ObjectId("5fb57d3202de9c7a2fb2e25b"),
        "items" : {
                "name" : "window_tint",
                "type" : [
                        "external",
                        "utility"
                ],
                "price" : NumberDecimal("300"),
                "quantity" : 4
        }
}
{
        "_id" : ObjectId("5fb57d3202de9c7a2fb2e25b"),
        "items" : {
                "name" : "seat_covers",
                "type" : [
                        "internal",
                        "accessory",
                        "utility"
                ],
                "price" : NumberDecimal("550"),
                "quantity" : 2
        }
}
mongos>
```

As you can see from the output documents, each item in all the "items" embedded arrays are deconstructed as individual items. You can further deconstruct the resulting data set using the "type" array, which we do in the next example.

Deconstruction syntax:

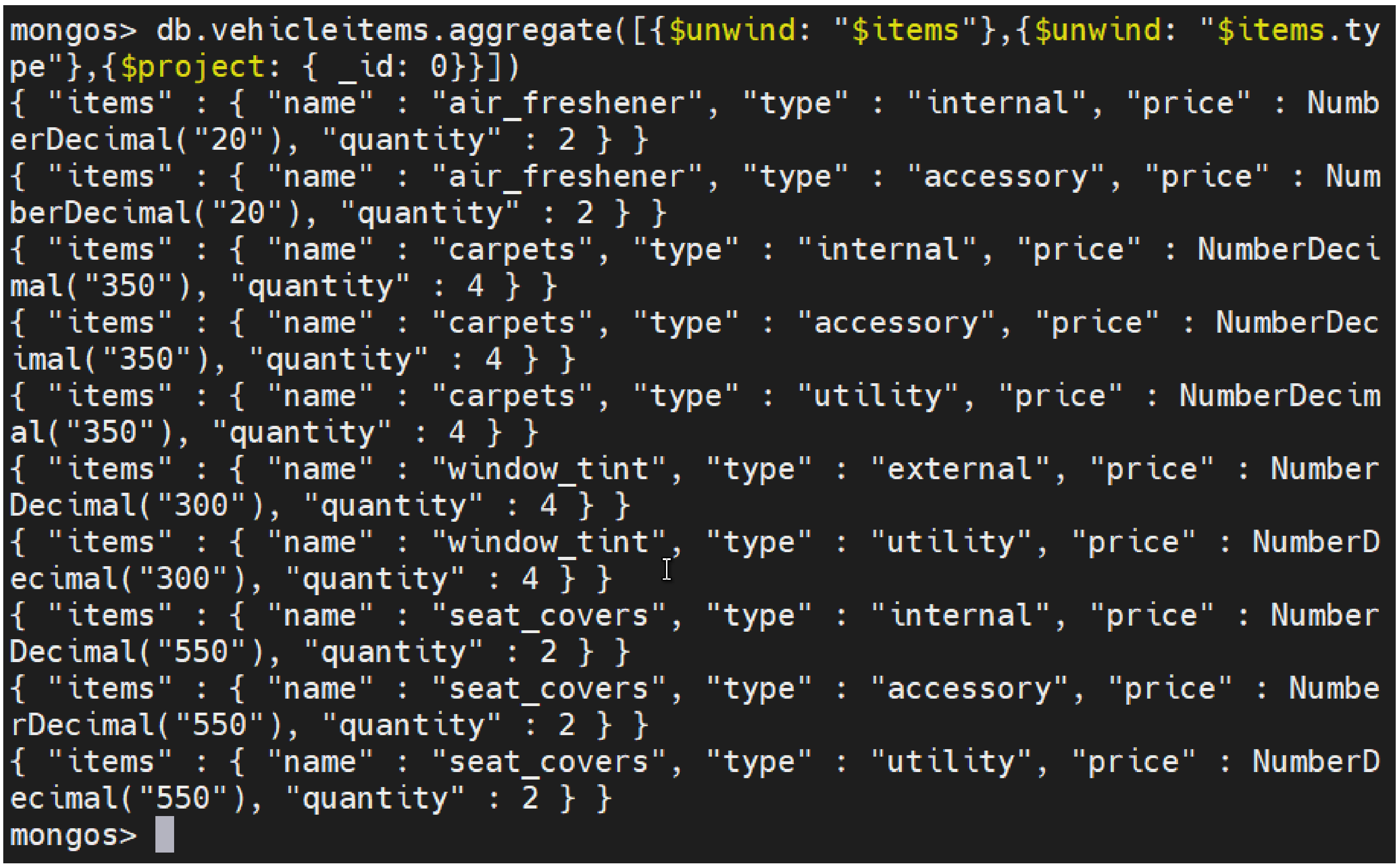
```
{$unwind: "$items"},{$unwind: "$items.type"}
```

Complete syntax:

```
db.vehicleitems.aggregate()
```

Use the projection operator to remove the “objectId” field for a cleaner output.

Result:



```
mongos> db.vehicleitems.aggregate([{$unwind: "$items"},{$unwind: "$items.ty
pe"},{$project: { _id: 0}}])
{ "items" : { "name" : "air_freshener", "type" : "internal", "price" : Numb
erDecimal("20"), "quantity" : 2 } }
{ "items" : { "name" : "air_freshener", "type" : "accessory", "price" : Num
berDecimal("20"), "quantity" : 2 } }
{ "items" : { "name" : "carpets", "type" : "internal", "price" : NumberDeci
mal("350"), "quantity" : 4 } }
{ "items" : { "name" : "carpets", "type" : "accessory", "price" : NumberDec
imal("350"), "quantity" : 4 } }
{ "items" : { "name" : "carpets", "type" : "utility", "price" : NumberDecim
al("350"), "quantity" : 4 } }
{ "items" : { "name" : "window_tint", "type" : "external", "price" : Number
Decimal("300"), "quantity" : 4 } }
{ "items" : { "name" : "window_tint", "type" : "utility", "price" : NumberD
ecimal("300"), "quantity" : 4 } }
{ "items" : { "name" : "seat_covers", "type" : "internal", "price" : Number
Decimal("550"), "quantity" : 2 } }
{ "items" : { "name" : "seat_covers", "type" : "accessory", "price" : Numbe
rDecimal("550"), "quantity" : 2 } }
{ "items" : { "name" : "seat_covers", "type" : "utility", "price" : NumberD
ecimal("550"), "quantity" : 2 } }
mongos>
```

That’s the end of this MongoDB $unwind tutorial.

Related reading

- BMC Machine Learning & Big Data Blog
- MongoDB Guide, a series of articles and tutorials
- MongoDB: The Mongo Shell & Basic Commands
- Data Storage Explained: Data Lake vs Warehouse vs Database