# MangoDB Class

#MangoDB Notes

## MapReduce And Some Useful Operators

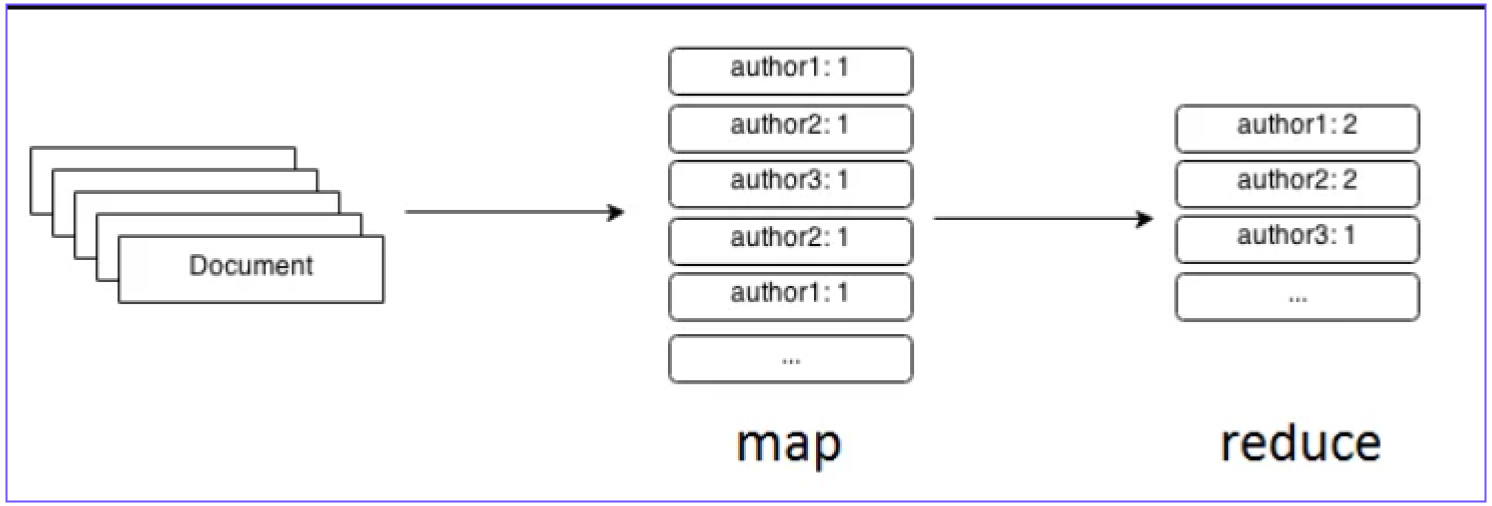
## What is MapReduce?

- In MongoDB, map-reduce is a data processing programming model that helps to perform operations on large data sets and produce aggregated results.
- MongoDB provides the mapReduce() function to perform the map-reduce operations. This function has two main functions, i.e., map function and reduce function.
- The map function is used to group all the data based on the key-value and the reduce function is used to perform operations on the mapped data.
- So, the data is independently mapped and reduced in different spaces and then combined together in the function and the result will save to the specified new collection.

- This mapReduce() function generally operated on large data sets only.
- Using Map Reduce you can perform aggregation operations such as max, avg on the data using some key and it is similar to groupBy in SQL. It performs on data independently and parallel.

## When to use Map-Reduce?

- In MongoDB, you can use Map-reduce when your aggregation query is slow because data is present in a large amount and the aggregation query is taking more time to process. So using map-reduce you can perform action faster than aggregation query.



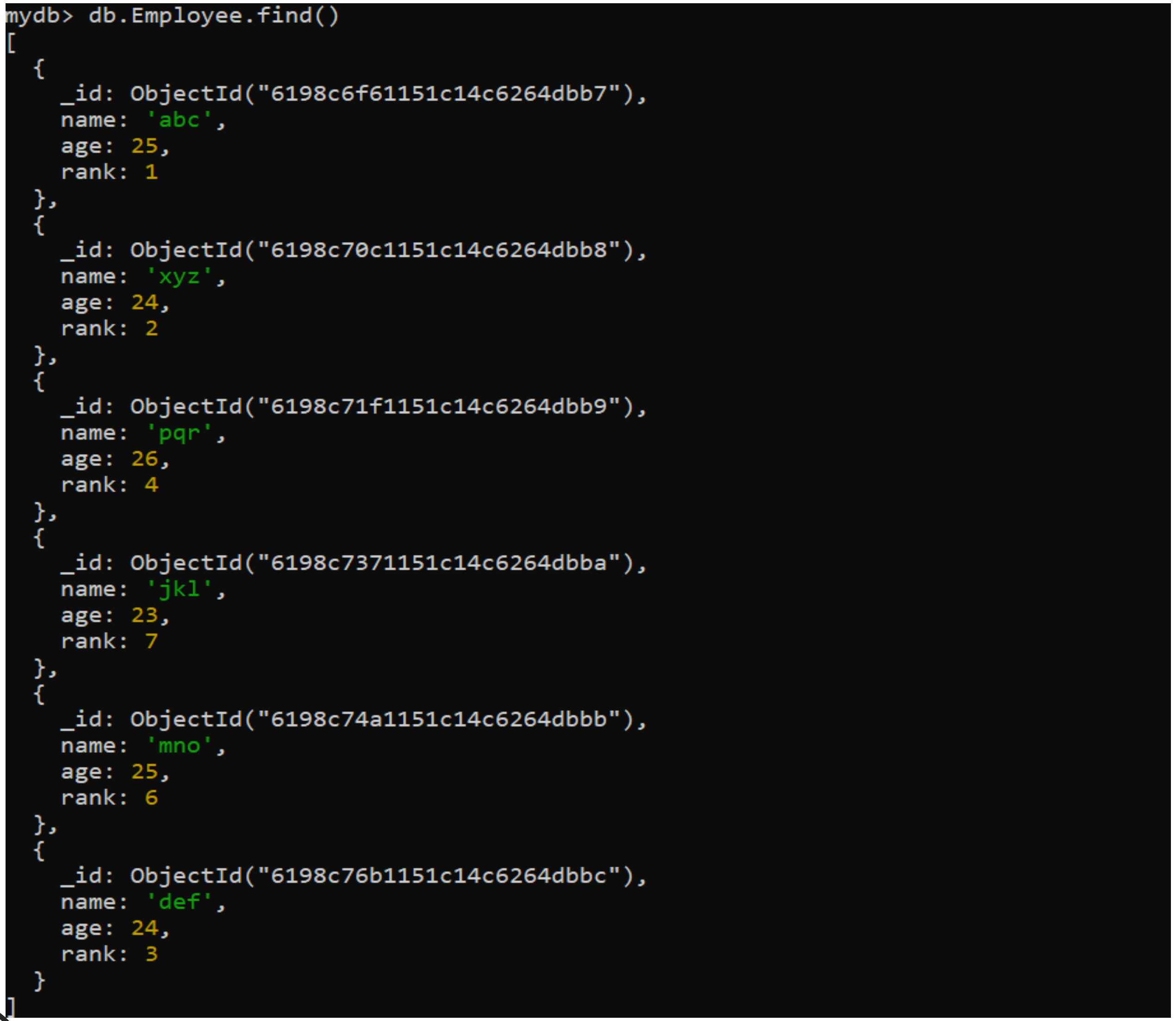
## Syntax

```
db.collection.mapReduce(
    function() {emit(key, value);},  //Define map function
   //Define reduce function
   function(key,values) {return reduceFunction}, {
             out: collection,
      query: document,
      sort: document,
      limit: number
   }
)
```

```
db.collectionName.mapReduce(
... map(),
...reduce(),
...query{},
...output{}
);
```

- The **map** is a javascript function that maps a value with a key and emits a key-value pair.
- The **reduce** is a javascript function that reduces or groups all the documents having the same key.
- The **out** specifies the location of the map-reduce query result.
- The **query** specifies the optional selection criteria for selecting documents.
- The **sort** specifies the optional sort criteria.
- The **limit** specifies the optional maximum number of documents to be returned.

# Employee Collection Data

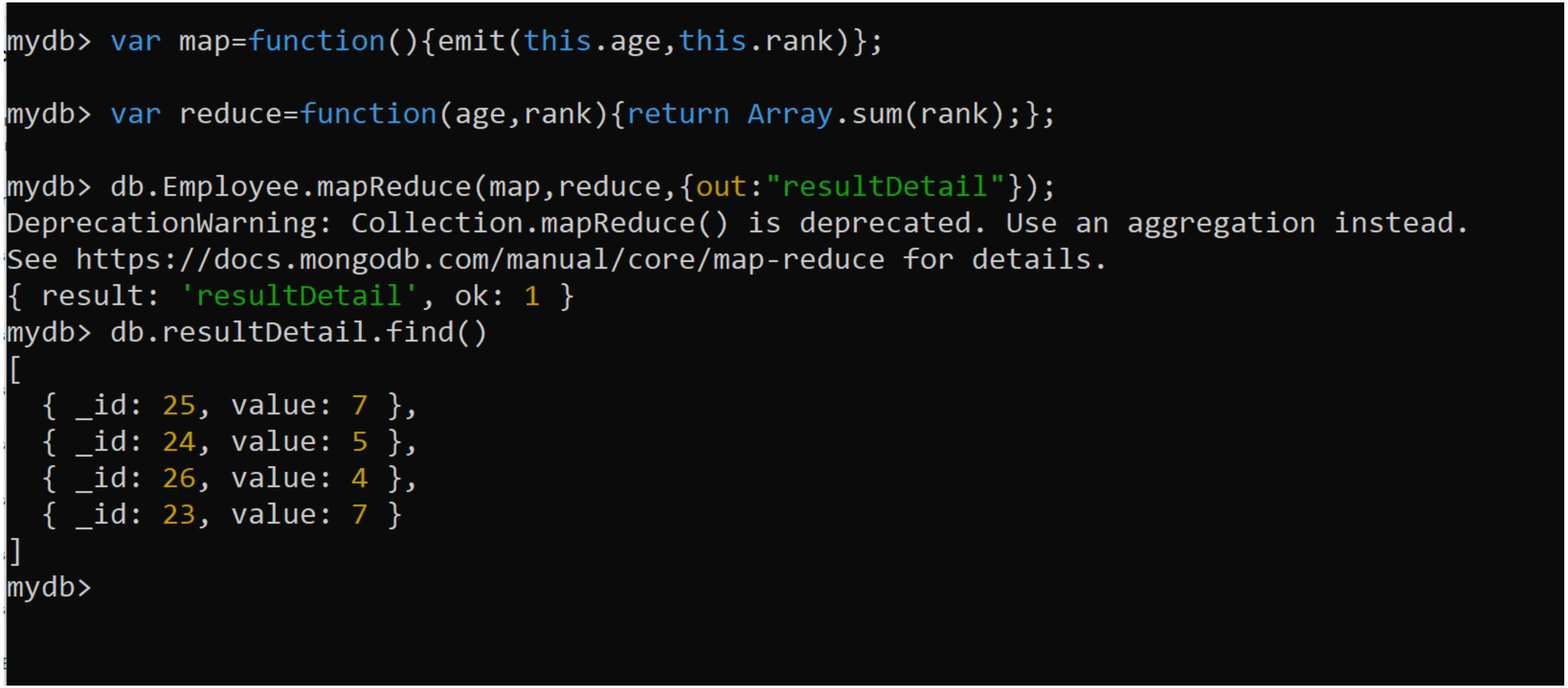


```
mydb> db.Employee.find()
[
  {
    _id: ObjectId("6198c6f61151c14c6264dbb7"),
    name: 'abc',
    age: 25,
    rank: 1
  },
  {
    _id: ObjectId("6198c70c1151c14c6264dbb8"),
    name: 'xyz',
    age: 24,
    rank: 2
  },
  {
    _id: ObjectId("6198c71f1151c14c6264dbb9"),
    name: 'pqr',
    age: 26,
    rank: 4
  },
  {
    _id: ObjectId("6198c7371151c14c6264dbba"),
    name: 'jkl',
    age: 23,
    rank: 7
  },
  {
    _id: ObjectId("6198c74a1151c14c6264dbbb"),
    name: 'mno',
    age: 25,
    rank: 6
  },
  {
    _id: ObjectId("6198c76b1151c14c6264dbbc"),
    name: 'def',
    age: 24,
    rank: 3
  }
]
```

- Find the sum of ranks grouped by ages:
- Define Map Function
  - We have defining map function to process each statement from the collection.
  - var map=function(){ emit(this.age,this.rank)};
- Define Reduce Function
  - We have defined reduce function to reduce the single object from all MongoDB MapReduce method values.
  - var reduce=function(age,rank){ return Array.sum(rank);};

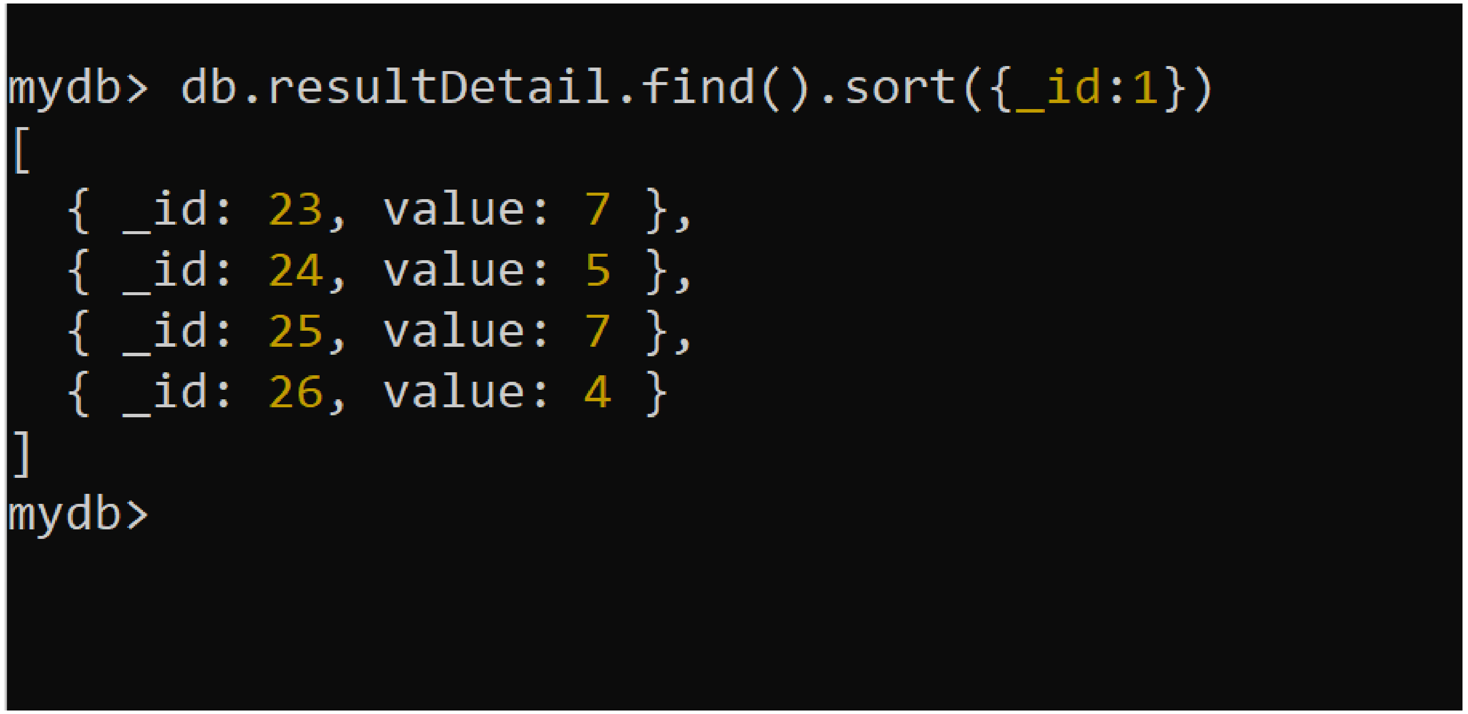
- Perform MapReduce Operation
  - After creating a map and reduce function we have performing MapReduce operation on Employee table are as follows.
  - db.employee.mapReduce(map,reduce,{out :"resultDetail"});
- Show resultDetail data
  - db.resultDetail.find()

## Example of mapReduce()



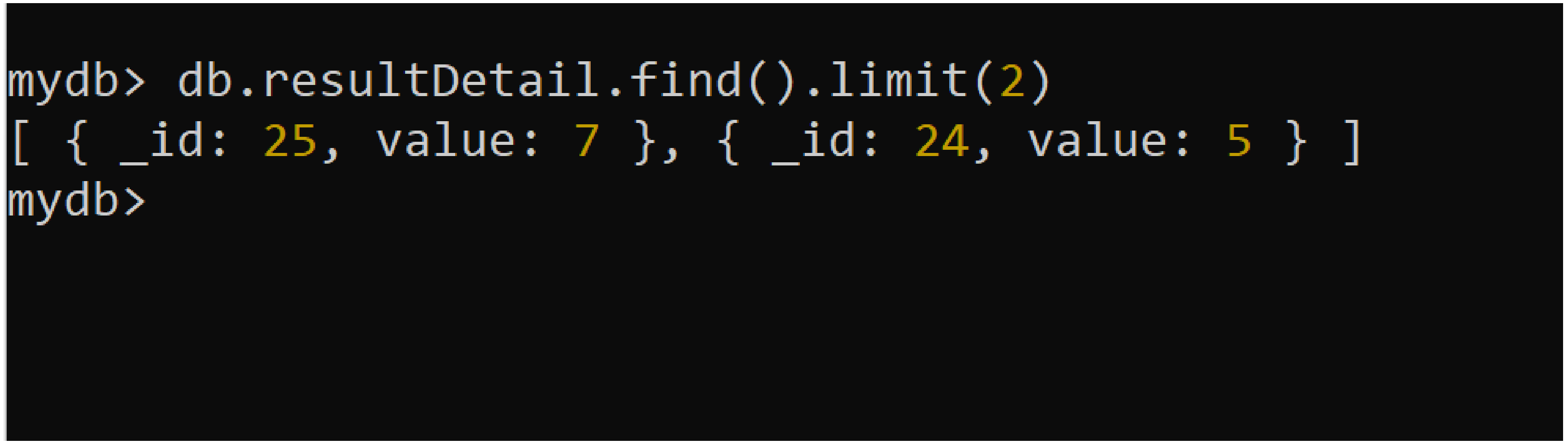
```
mydb> var map=function(){emit(this.age,this.rank)};

mydb> var reduce=function(age,rank){return Array.sum(rank);};

mydb> db.Employee.mapReduce(map,reduce,{out:"resultDetail"});
DeprecationWarning: Collection.mapReduce() is deprecated. Use an aggregation instead.
See https://docs.mongodb.com/manual/core/map-reduce for details.
{ result: 'resultDetail', ok: 1 }
mydb> db.resultDetail.find()
[
  { _id: 25, value: 7 },
  { _id: 24, value: 5 },
  { _id: 26, value: 4 },
  { _id: 23, value: 7 }
]
mydb>
```

## Result Using Sort



```
mydb> db.resultDetail.find().sort({_id:1})
[
  { _id: 23, value: 7 },
  { _id: 24, value: 5 },
  { _id: 25, value: 7 },
  { _id: 26, value: 4 }
]
mydb>
```

## Limit on MapReduce Command

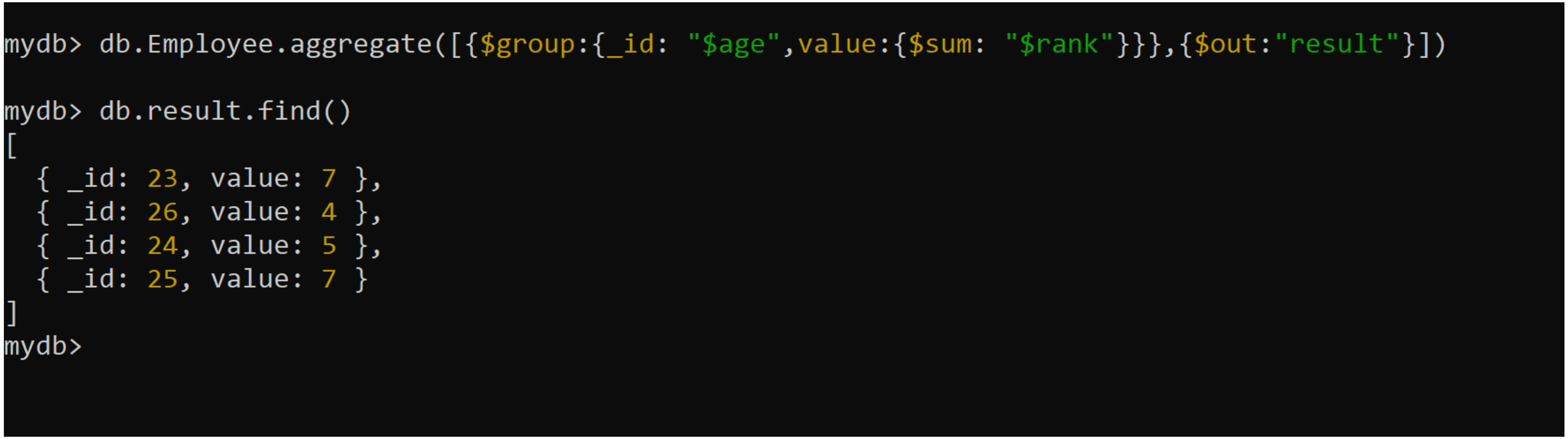


```
mydb> db.resultDetail.find().limit(2)
[ { _id: 25, value: 7 }, { _id: 24, value: 5 } ]
mydb>
```

## Aggregation Alternative

- Using the available aggregation pipeline operators, you can rewrite the map-reduce operation without defining custom functions:

## Example



```
mydb> db.Employee.aggregate([{$group:{_id: "$age",value:{$sum: "$rank"}}},{$out:"result"}])

mydb> db.result.find()
[
  { _id: 23, value: 7 },
  { _id: 26, value: 4 },
  { _id: 24, value: 5 },
  { _id: 25, value: 7 }
]
mydb>
```

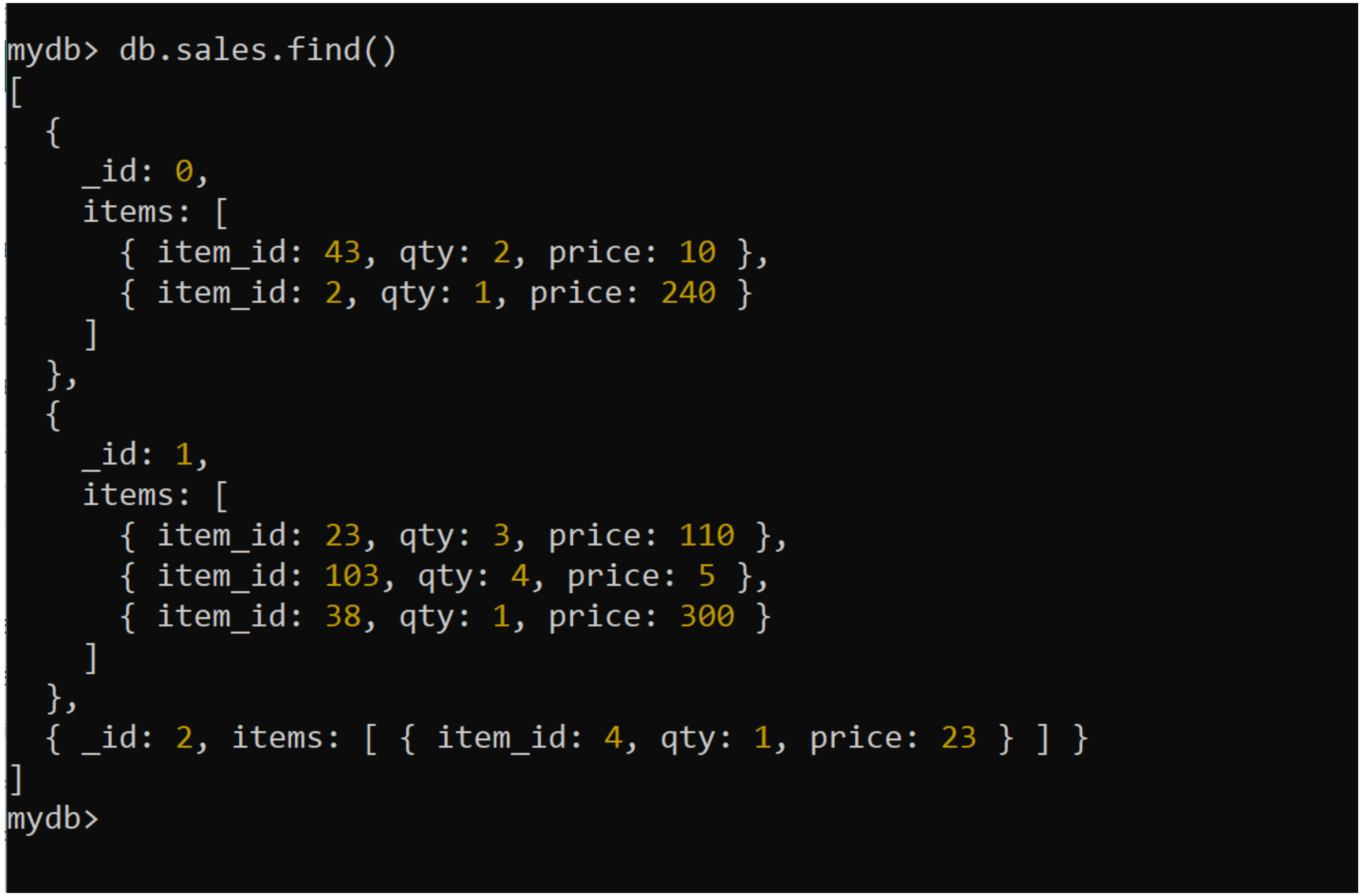
# $filter (aggregation)

- Selects a subset of an array to return based on the specified condition. Returns an array with only those elements that match the condition.
- The returned elements are in the original order.
- $filter operator uses three variables:
  - **input –** This represents the array that we want to extract.
  - **cond –** This represents the set of conditions that must be met.
  - **as –** This optional field contains a name for the variable that represent each element of the input array.

- **Syntax:**

  { $filter: { input: <array>, as: <string>, cond: <expression> } }

# Sales Collection Data

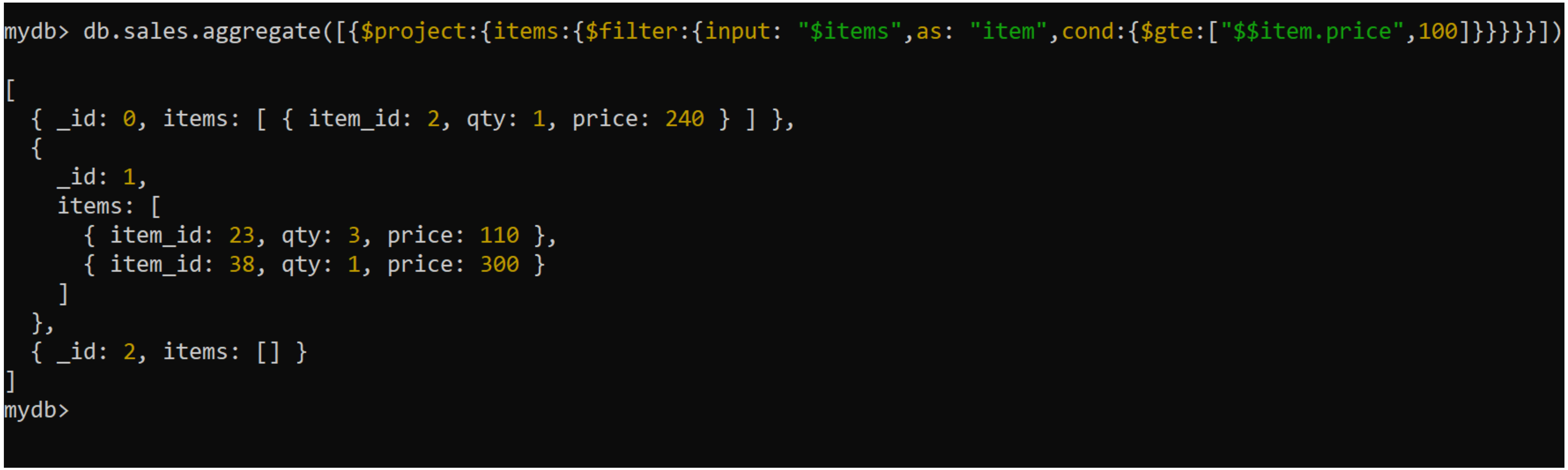


```
mydb> db.sales.find()
[
  {
    _id: 0,
    items: [
      { item_id: 43, qty: 2, price: 10 },
      { item_id: 2, qty: 1, price: 240 }
    ]
  },
  {
    _id: 1,
    items: [
      { item_id: 23, qty: 3, price: 110 },
      { item_id: 103, qty: 4, price: 5 },
      { item_id: 38, qty: 1, price: 300 }
    ]
  },
  { _id: 2, items: [ { item_id: 4, qty: 1, price: 23 } ] }
]
mydb>
```

## Example

- The following example filters the items array to only include documents that have a price greater than or equal to 100.

```
db.sales.aggregate([
  {
    $project: {
      items: {
        $filter: {
          input: "$items",
          as: "item",
          cond: { $gte: [ "$$item.price", 100 ] }
        }
      }
    }
  }
])
```



```
mydb> db.sales.aggregate([{$project:{items:{$filter:{input: "$items",as: "item",cond:{$gte:["$$item.price",100]}}}}}])

[
  { _id: 0, items: [ { item_id: 2, qty: 1, price: 240 } ] },
  {
    _id: 1,
    items: [
      { item_id: 23, qty: 3, price: 110 },
      { item_id: 38, qty: 1, price: 300 }
    ]
  },
  { _id: 2, items: [] }
]
mydb>
```

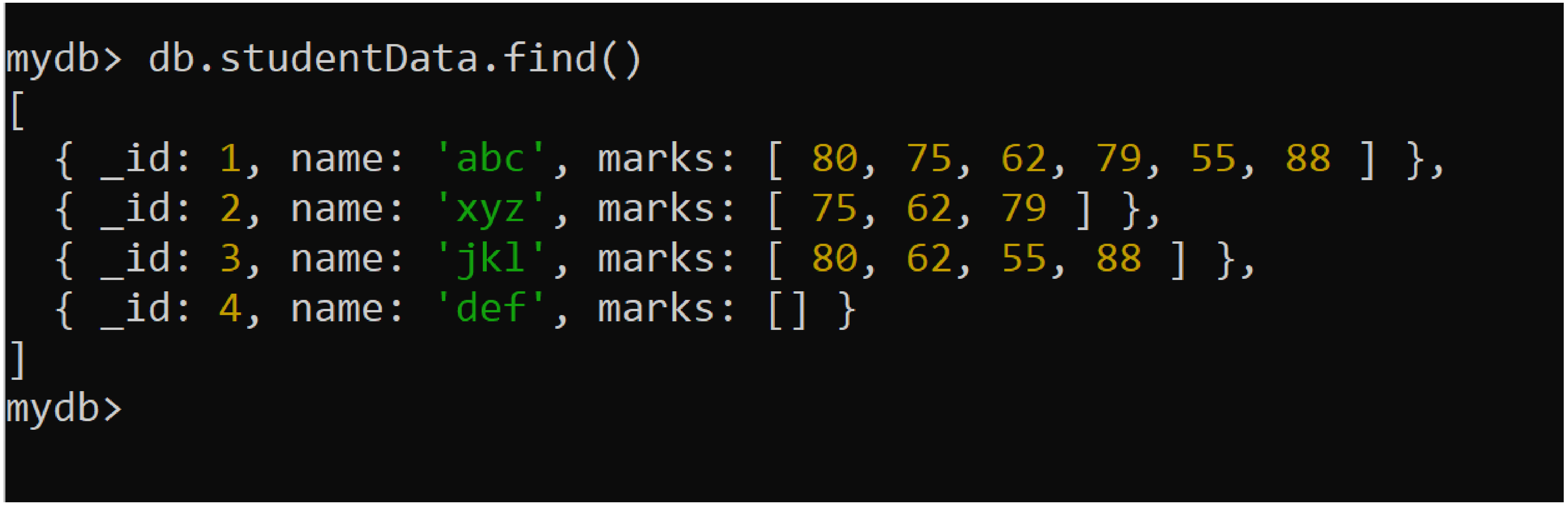
## $size

- The $size operator matches any array with the number of elements specified by the argument.

- **Syntax :-**

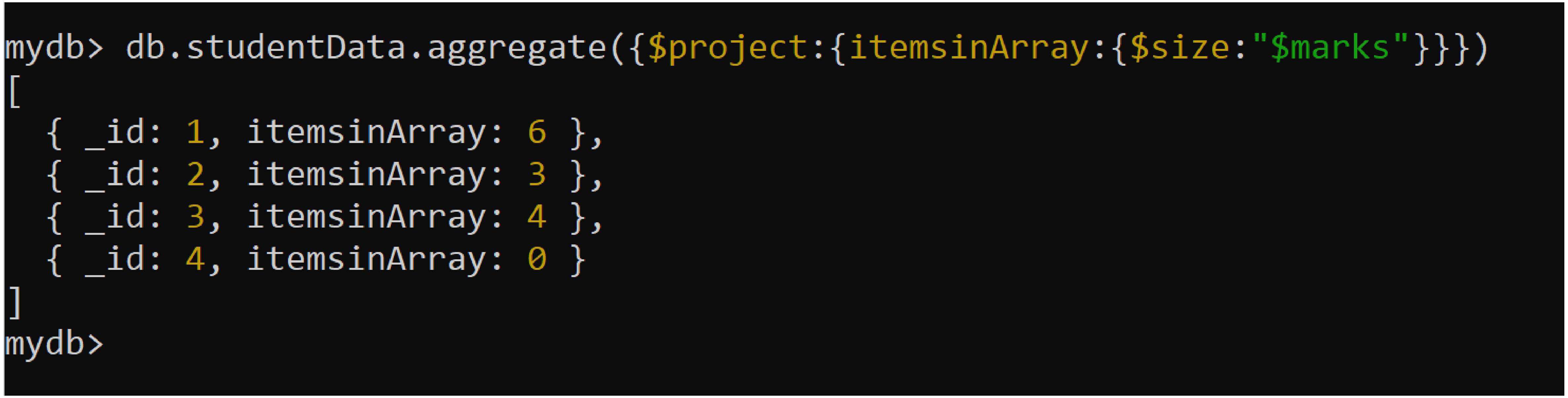
```
db.collection.find( { field: { $size: <expression> } } );
```

# studentData Collection Data



```
mydb> db.studentData.find()
[
  { _id: 1, name: 'abc', marks: [ 80, 75, 62, 79, 55, 88 ] },
  { _id: 2, name: 'xyz', marks: [ 75, 62, 79 ] },
  { _id: 3, name: 'jkl', marks: [ 80, 62, 55, 88 ] },
  { _id: 4, name: 'def', marks: [] }
]
mydb>
```

## $size Example

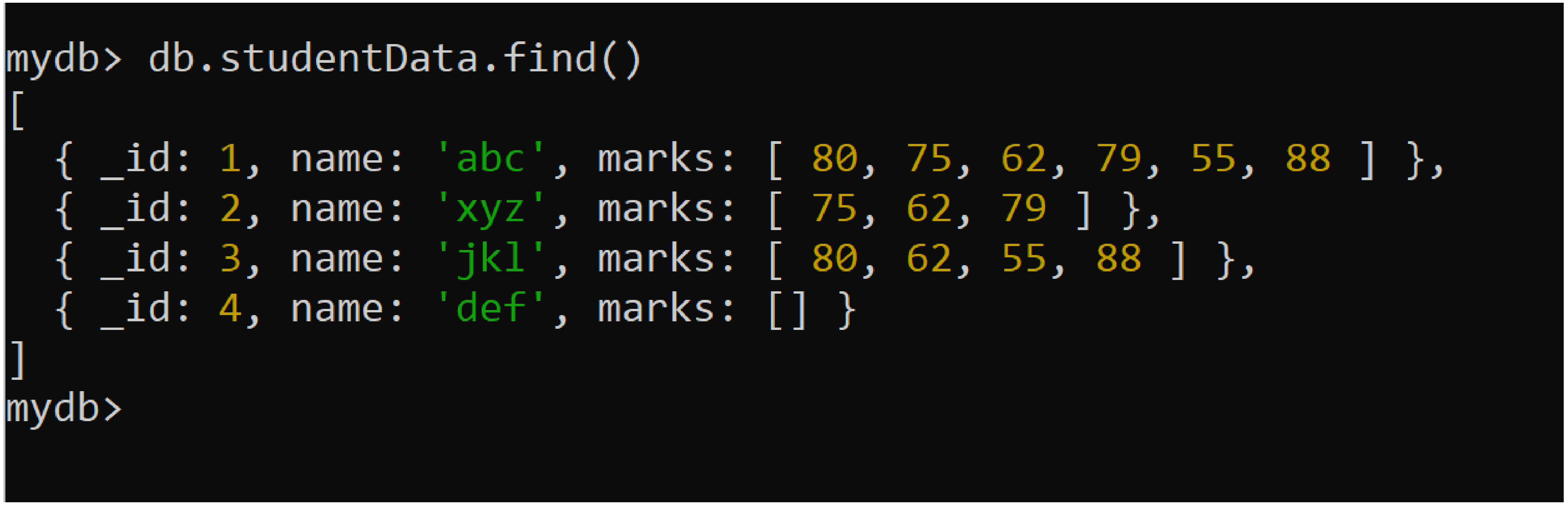

```
mydb> db.studentData.aggregate({$project:{itemsinArray:{$size:"$marks"}}})
[
  { _id: 1, itemsinArray: 6 },
  { _id: 2, itemsinArray: 3 },
  { _id: 3, itemsinArray: 4 },
  { _id: 4, itemsinArray: 0 }
]
mydb>
```

## $arrayElemAt

- Returns the element at the specified array index.

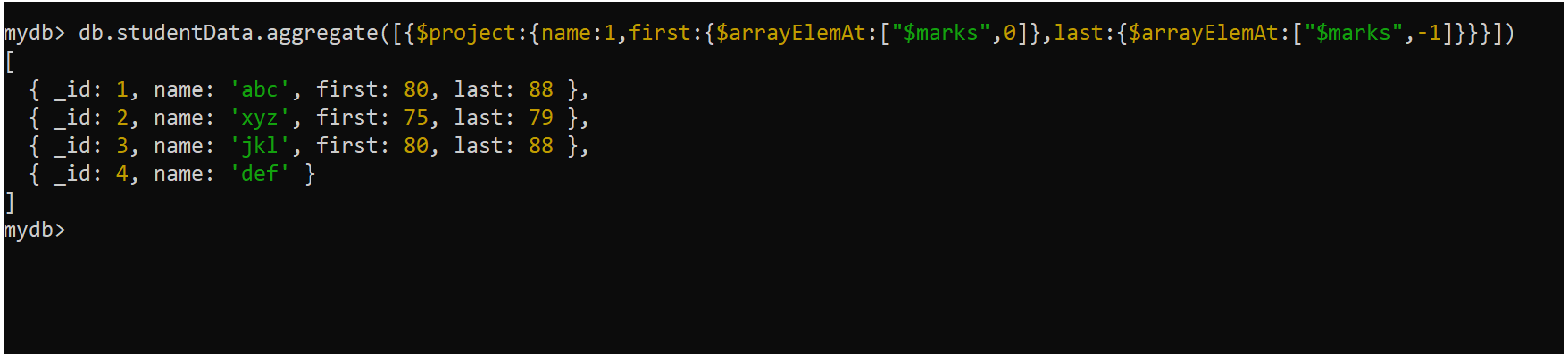
- **Syntax:**
  - { $arrayElemAt: [ <array>, <idx> ] }
    - The <array> expression can be any valid expression that resolves to an array.
    - The <idx> expression can be any valid expression that resolves to an integer:
      - If the <idx> expression resolves to zero or a positive integer, $arrayElemAt returns the element at the idx position, counting from the start of the array.
      - If the <idx> expression resolves to a negative integer, $arrayElemAt returns the element at the idx position, counting from the end of the array.

## studentData Collection Data



```
mydb> db.studentData.find()
[
  { _id: 1, name: 'abc', marks: [ 80, 75, 62, 79, 55, 88 ] },
  { _id: 2, name: 'xyz', marks: [ 75, 62, 79 ] },
  { _id: 3, name: 'jkl', marks: [ 80, 62, 55, 88 ] },
  { _id: 4, name: 'def', marks: [] }
]
mydb>
```

# $arrayElemAt Example



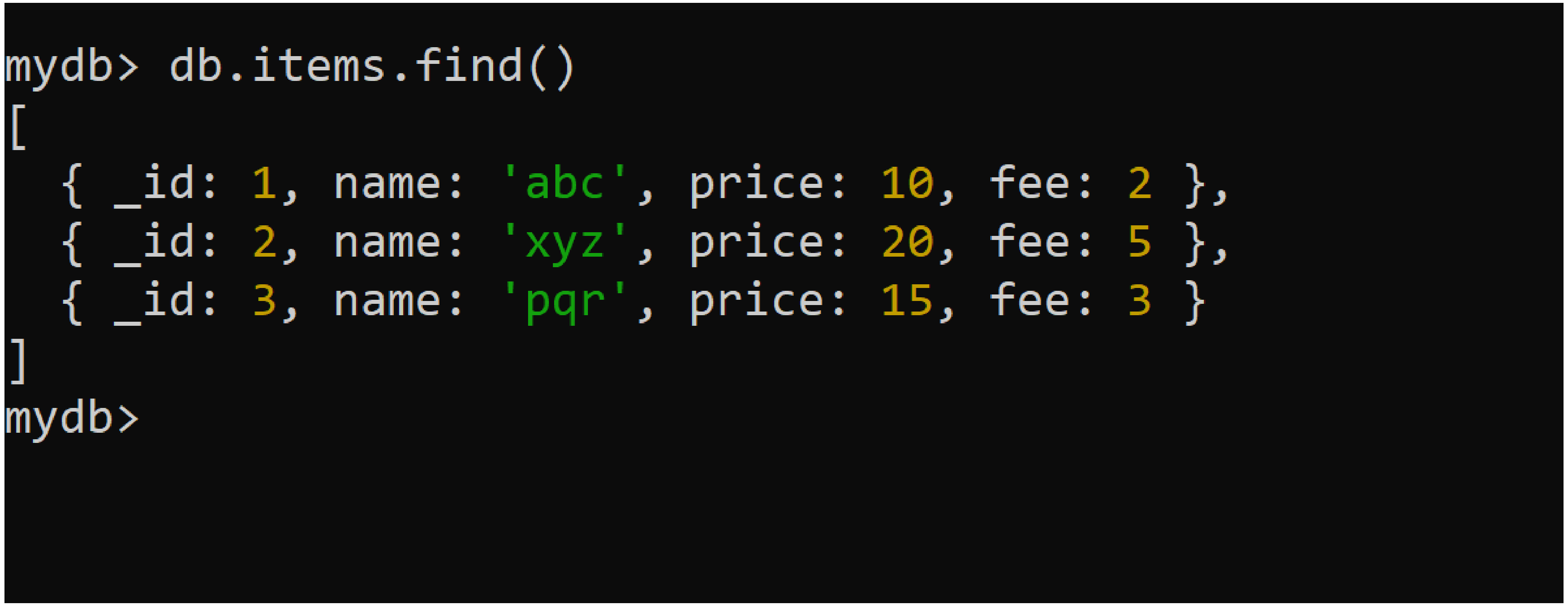
```
mydb> db.studentData.aggregate([{$project:{name:1,first:{$arrayElemAt:["$marks",0]},last:{$arrayElemAt:["$marks",-1]}}}])
[
  { _id: 1, name: 'abc', first: 80, last: 88 },
  { _id: 2, name: 'xyz', first: 75, last: 79 },
  { _id: 3, name: 'jkl', first: 80, last: 88 },
  { _id: 4, name: 'def' }
]
mydb>
```

## $add

- Adds numbers together or adds numbers and a date.
- (Add 2 Field Value and Display)

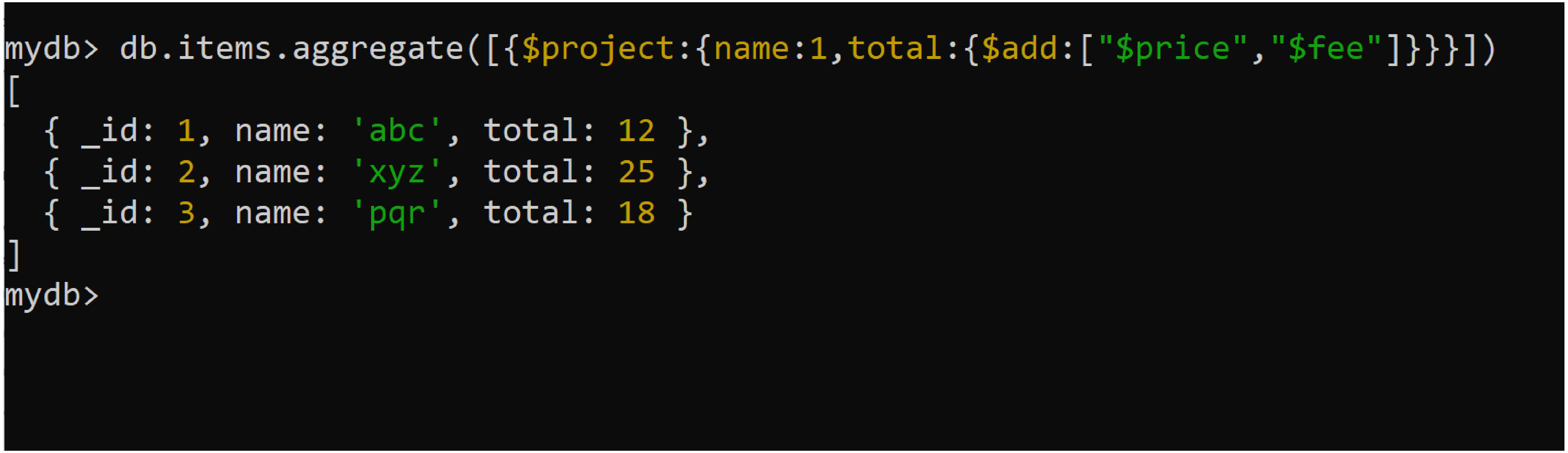
- **Syntax:**
  **{ $add: [ <expression1>, <expression2>, ... ] }**

## Items Collection Data



```
mydb> db.items.find()
[
  { _id: 1, name: 'abc', price: 10, fee: 2 },
  { _id: 2, name: 'xyz', price: 20, fee: 5 },
  { _id: 3, name: 'pqr', price: 15, fee: 3 }
]
mydb>
```

## $add Example (Sum of 2 Fields)



```
mydb> db.items.aggregate([{$project:{name:1,total:{$add:["$price","$fee"]}}}])
[
  { _id: 1, name: 'abc', total: 12 },
  { _id: 2, name: 'xyz', total: 25 },
  { _id: 3, name: 'pqr', total: 18 }
]
mydb>
```

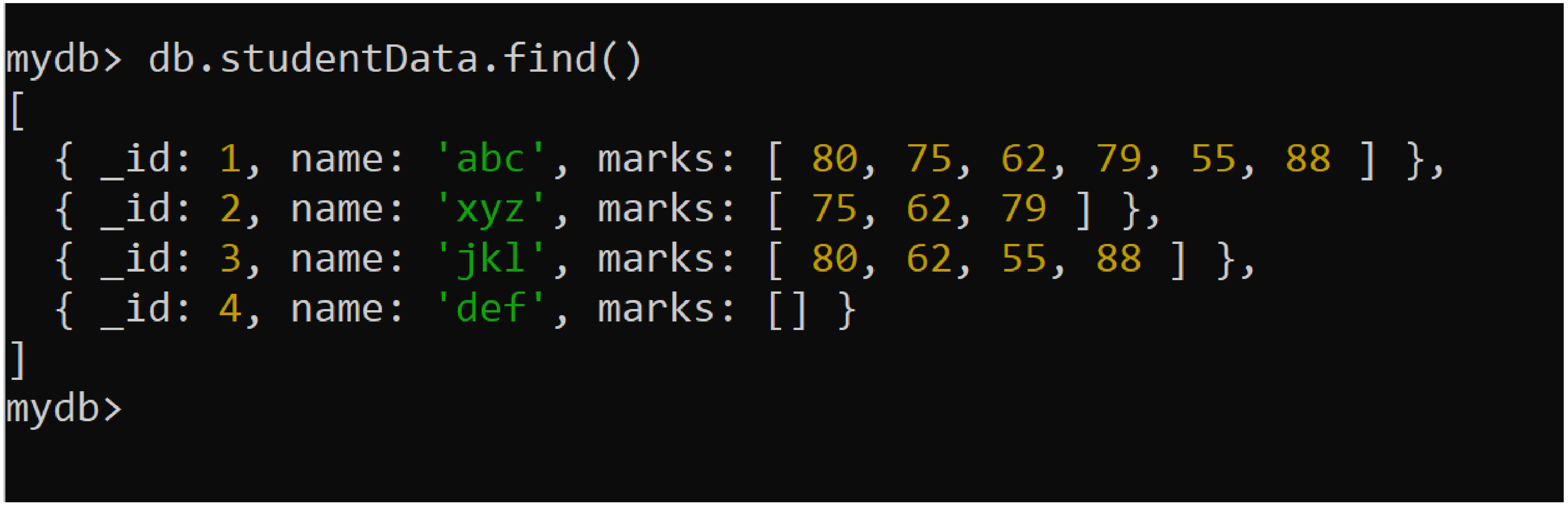
## $map

- Applies an expression to each item in an array and returns an array with the applied results.
- **Syntax:**
- { $map: { input: <expression>, as: <string>, in: <expression> } }
  - input
    - An expression that resolves to an array.
  - as
    - Optional. A name for the variable that represents each individual element of the input array. If no name is specified, the variable name defaults to this.
  - in
    - An expression that is applied to each element of the input array. The expression references each element individually with the variable name specified in as.

## $map Example

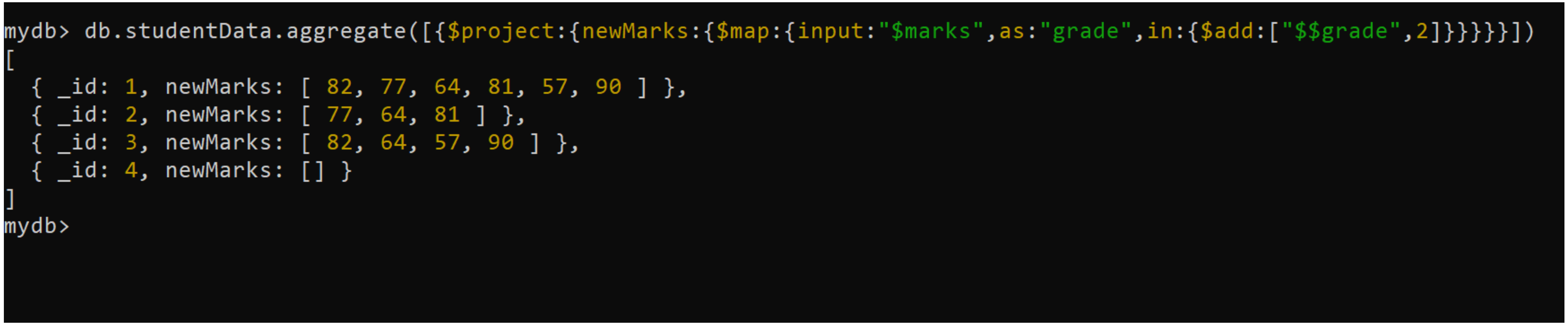
```
db.studentData.aggregate(
  [     { $project:
      { newMarks:
        {     $map:
            { input: "$marks",
              as: "grade",
              in: { $add: [ "$$grade", 2 ] }
            }
        }
      }
    }
  ]
)
```

## studentData Collection Data



```
mydb> db.studentData.find()
[
  { _id: 1, name: 'abc', marks: [ 80, 75, 62, 79, 55, 88 ] },
  { _id: 2, name: 'xyz', marks: [ 75, 62, 79 ] },
  { _id: 3, name: 'jkl', marks: [ 80, 62, 55, 88 ] },
  { _id: 4, name: 'def', marks: [] }
]
mydb>
```

## Add 2 Value in All Array Value



```
mydb> db.studentData.aggregate([{$project:{newMarks:{$map:{input:"$marks",as:"grade",in:{$add:["$$grade",2]}}}}}])
[
  { _id: 1, newMarks: [ 82, 77, 64, 81, 57, 90 ] },
  { _id: 2, newMarks: [ 77, 64, 81 ] },
  { _id: 3, newMarks: [ 82, 64, 57, 90 ] },
  { _id: 4, newMarks: [] }
]
mydb>
```

## $switch

- Evaluates a series of case expressions.
- When it finds an expression which evaluates to true, $switch executes a specified expression and breaks out of the control flow.

- **Syntax:**

```
$switch: {
  branches: [
    { case: <expression>, then: <expression> },
    { case: <expression>, then: <expression> },
    ...
  ],
  default: <expression>
}
```

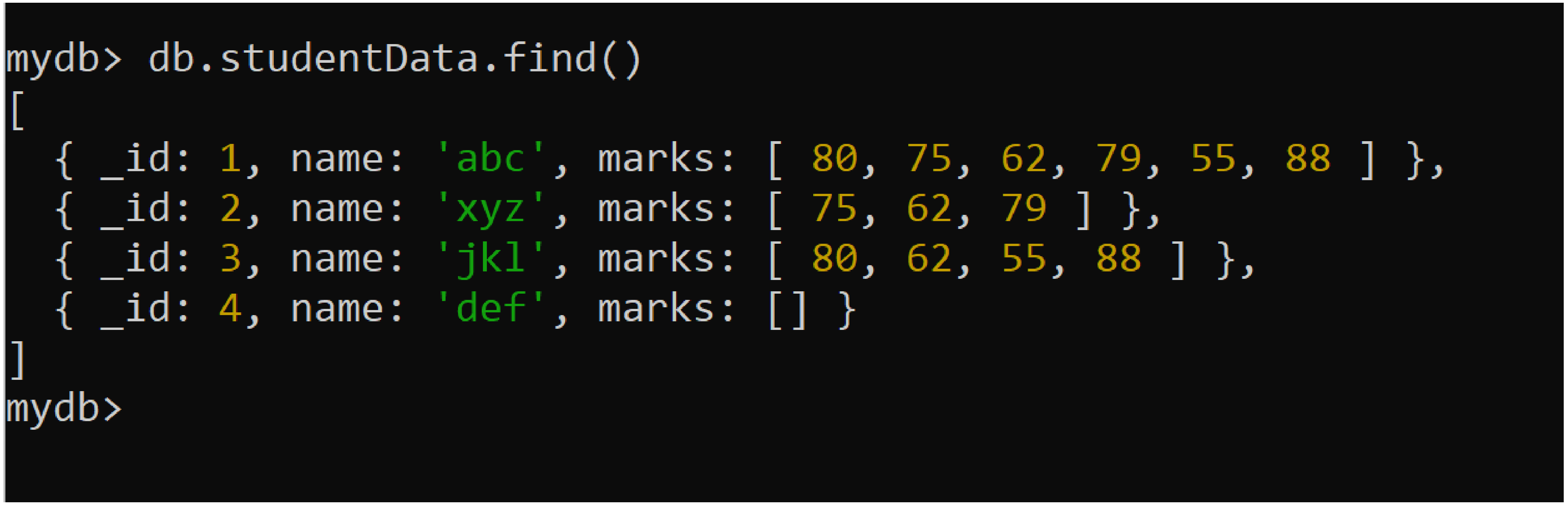
- The objects in the branches array must contain only a case field and a then field.

- branches
  - An array of control branch documents. Each branch is a document with the following fields:
- case
  - Can be any valid expression that resolves to a boolean.
- then
  - Can be any valid expression.
  - The branches array must contain at least one branch document.
- default
  - Optional. The path to take if no branch case expression evaluates to true.

## Example

```
db.studentData.aggregate( [
 {   $project:
     {
       "name" : 1,
       "summary" :
       {    $switch:
          {    branches: [
             {
               case: { $gte : [ { $avg : "$marks" }, 70 ] },
               then: "Doing great!"
             },
             {
               case: { $lt : [ { $avg : "$marks" }, 60 ] },
               then: "Needs improvement."
             }
            ],
            default: "No scores found."
          }
      } }  }] )
```

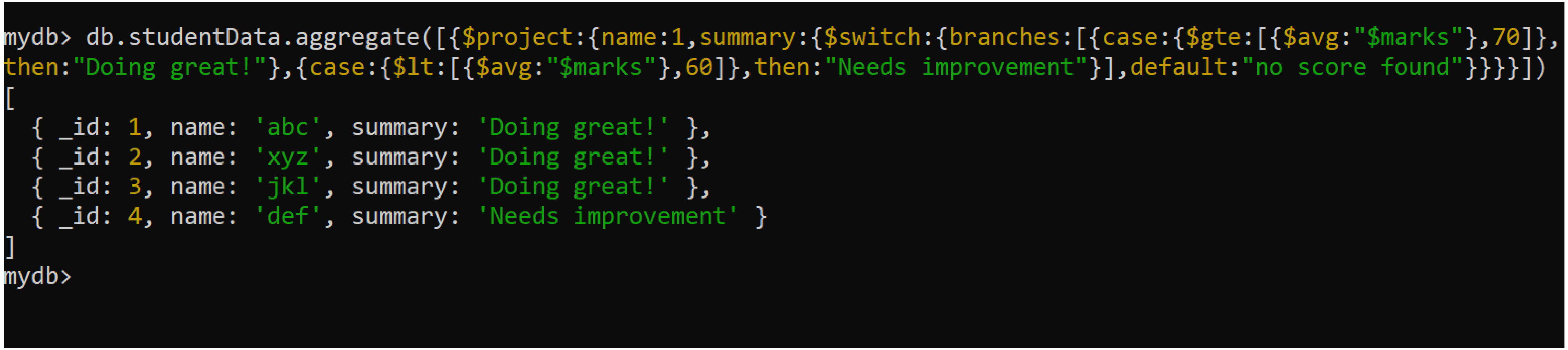
## studentData Collection Data



```
mydb> db.studentData.find()
[
  { _id: 1, name: 'abc', marks: [ 80, 75, 62, 79, 55, 88 ] },
  { _id: 2, name: 'xyz', marks: [ 75, 62, 79 ] },
  { _id: 3, name: 'jkl', marks: [ 80, 62, 55, 88 ] },
  { _id: 4, name: 'def', marks: [] }
]
mydb>
```

# Switch Condition

Marks > 70 = Doing Great
Marks < 60 = Needs Improvement
Else No Score Found



```
mydb> db.studentData.aggregate([{$project:{name:1,summary:{$switch:{branches:[{case:{$gte:[{$avg:"$marks"},70]},
then:"Doing great!"},{case:{$lt:[{$avg:"$marks"},60]},then:"Needs improvement"}],default:"no score found"}}}}])
[
  { _id: 1, name: 'abc', summary: 'Doing great!' },
  { _id: 2, name: 'xyz', summary: 'Doing great!' },
  { _id: 3, name: 'jkl', summary: 'Doing great!' },
  { _id: 4, name: 'def', summary: 'Needs improvement' }
]
mydb>
```

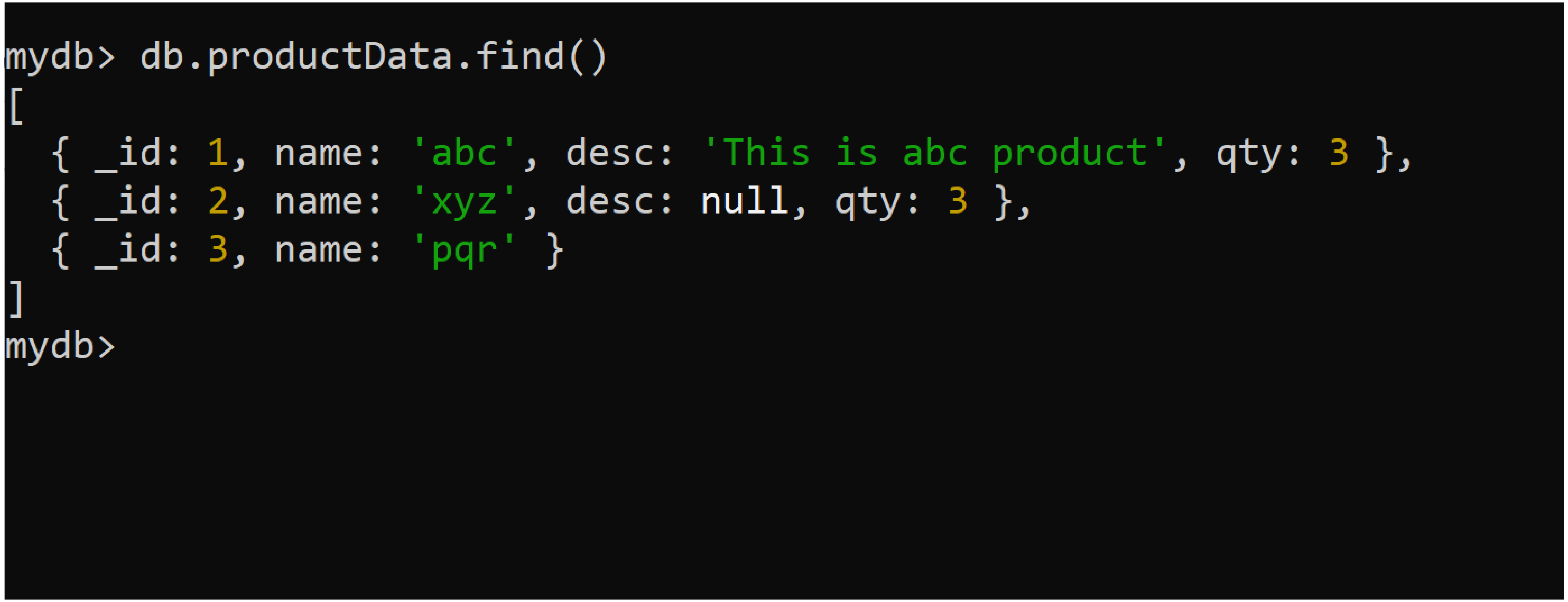
## $ifNull

- The $ifNull expression evaluates input expressions for null values and returns:
- $ifNull treats undefined values and missing fields as null.

- **Syntax :-**

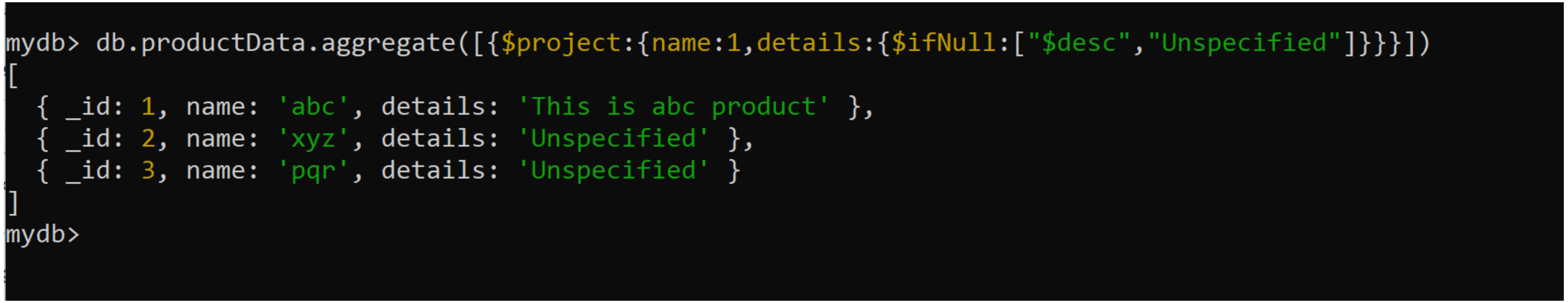
```
{
  $ifNull: [
    <input-expression>, <replacement-expression-if-null>
  ]
}
```

## productData Collection Data



```
mydb> db.productData.find()
[
  { _id: 1, name: 'abc', desc: 'This is abc product', qty: 3 },
  { _id: 2, name: 'xyz', desc: null, qty: 3 },
  { _id: 3, name: 'pqr' }
]
mydb>
```

# $ifNull Example (Null Check and Replace)



```
mydb> db.productData.aggregate([{$project:{name:1,details:{$ifNull:["$desc","Unspecified"]}}}])
[
  { _id: 1, name: 'abc', details: 'This is abc product' },
  { _id: 2, name: 'xyz', details: 'Unspecified' },
  { _id: 3, name: 'pqr', details: 'Unspecified' }
]
mydb>
```

## $ifNull Example

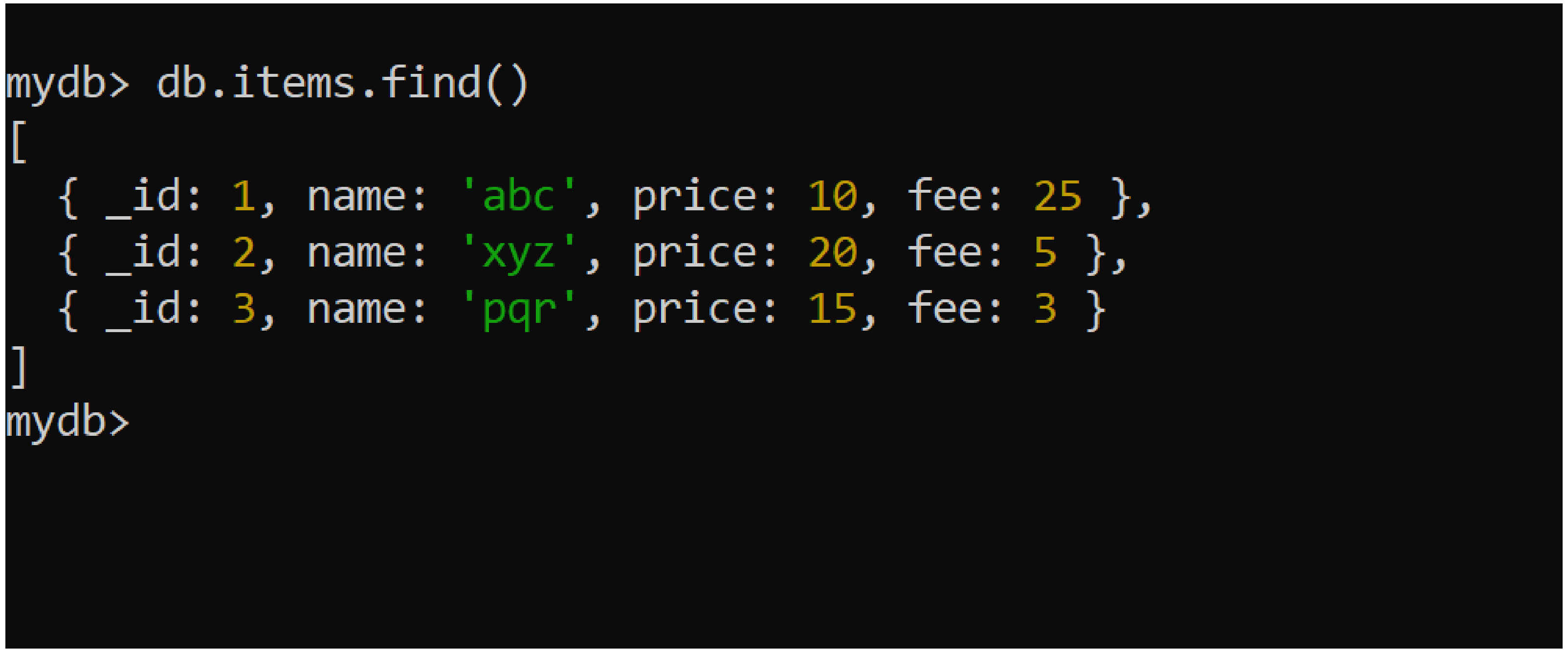
```
db.productData.aggregate(
  [
    {
      $project: {
        name: 1,
        details: { $ifNull: [ "$desc", "Unspecified" ] }
      }
    }
  ]
)
```

## $expr

- Allows the use of aggregation expressions within the query language.

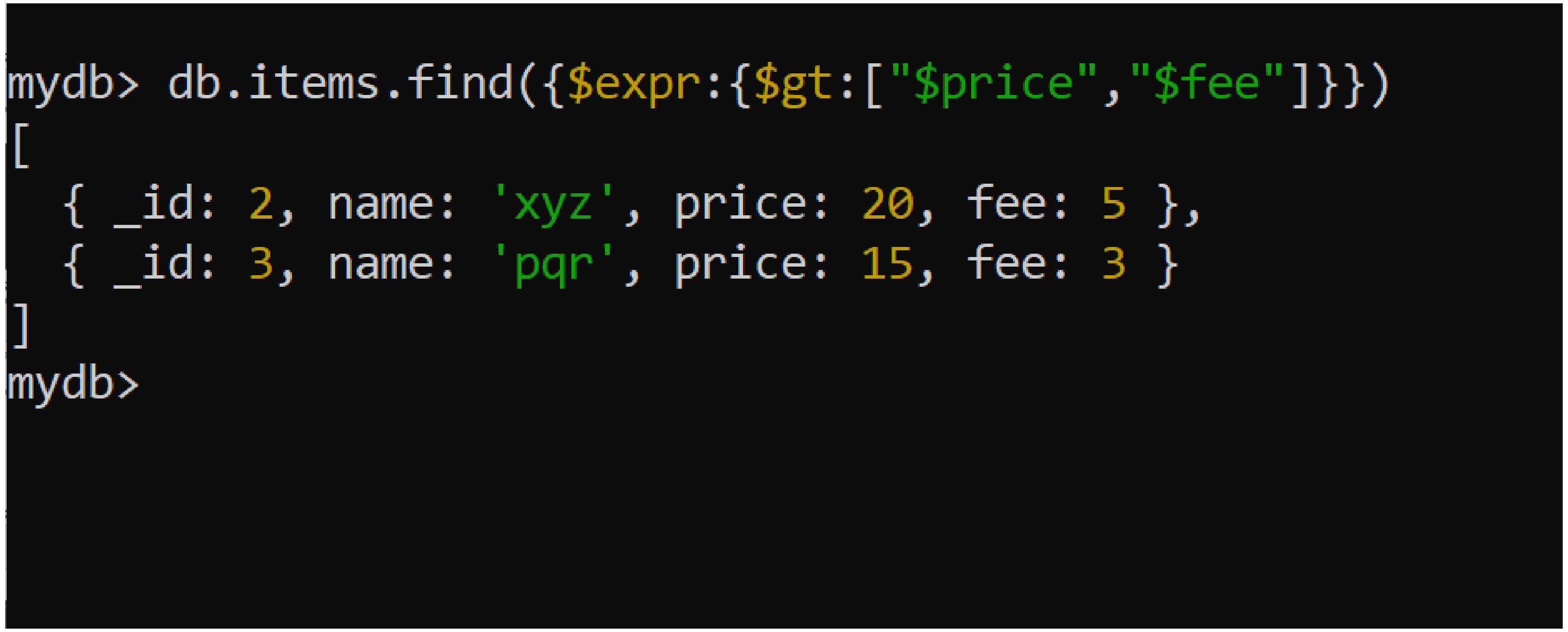
- **Syntax:**
  - **{ $expr: { <expression> } }**

## Items Collection Data


```
mydb> db.items.find()
[
  { _id: 1, name: 'abc', price: 10, fee: 25 },
  { _id: 2, name: 'xyz', price: 20, fee: 5 },
  { _id: 3, name: 'pqr', price: 15, fee: 3 }
]
mydb>
```

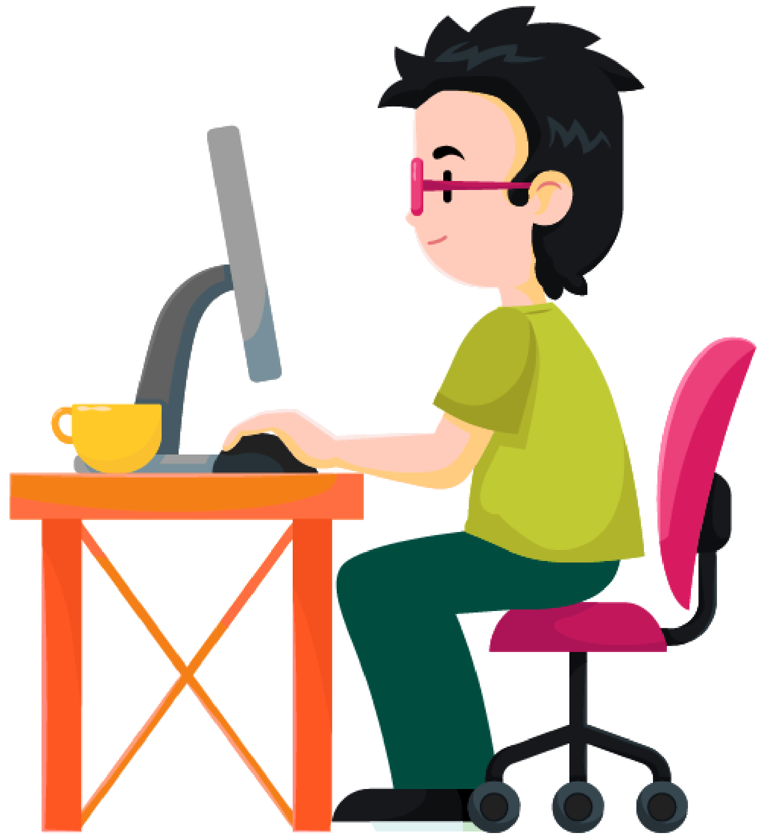
## $expr Example

- The following operation uses $expr to find documents where the price amount exceeds the fee:



```
mydb> db.items.find({$expr:{$gt:["$price","$fee"]}})
[
  { _id: 2, name: 'xyz', price: 20, fee: 5 },
  { _id: 3, name: 'pqr', price: 15, fee: 3 }
]
mydb>
```

Get Exclusive
Video Tutorials



www.aptutorials.com

https://www.youtube.com/user/Akashtips

Get More Details

www.akashsir.com

## If You Liked It !

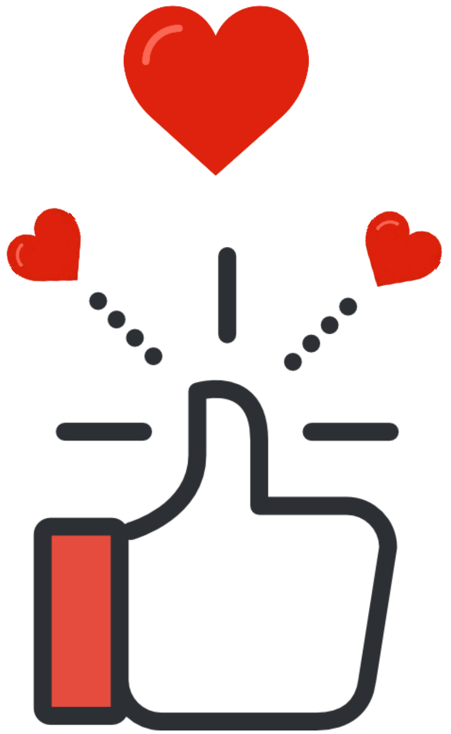
### Rating Us Now

**Just Dial**

https://www.justdial.com/Ahmedabad/Akash-Technolabs-Navrangpura-Bus-Stop-Navrangpura/079PXX79-XX79-170615221520-S5C4_BZDET

**Sulekha**

https://www.sulekha.com/akash-technolabs-navrangpura-ahmedabad-contact-address/ahmedabad



## Connect With Me

Akash Padhiyar
#AkashSir

www.akashsir.com
www.akashtechnolabs.com
www.akashpadhiyar.com
www.aptutorials.com

## # Social Info

Akash.padhiyar
Akashpadhiyar
Akash_padhiyar
+91 99786-21654

#Akashpadhiyar
#aptutorials