PROJECTION,LIMIT AND SELECTORS:

PROJECTION

MongoDB projection is a powerful tool that can be used to extract only the fields you need from a document—not all fields. It enables you to:

Project concise and transparent data

Filter the data set without impacting the overall database performance

Because MongoDB projection is built on top of the existing find() method, you can use any projection query without significant modifications to the existing functions. Plus, projection is a key factor when finding user-specific data from a given data set.

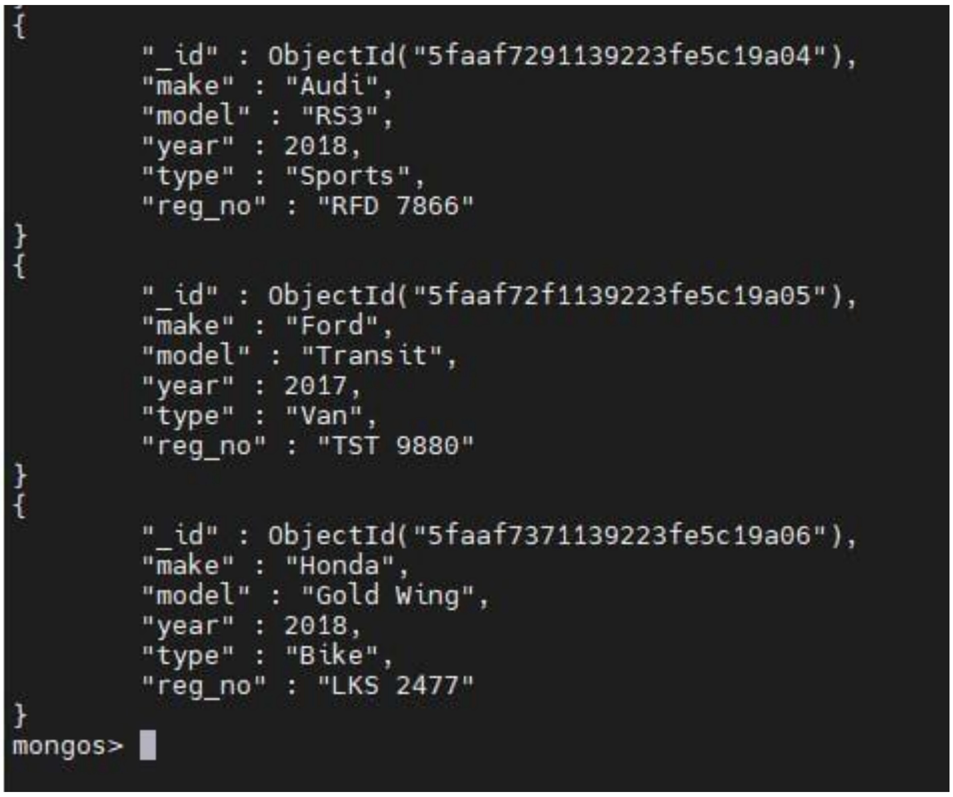
Basic Syntax of MongoDB Projection

```
db.collection_name.find({},{<field> : <value>})
```

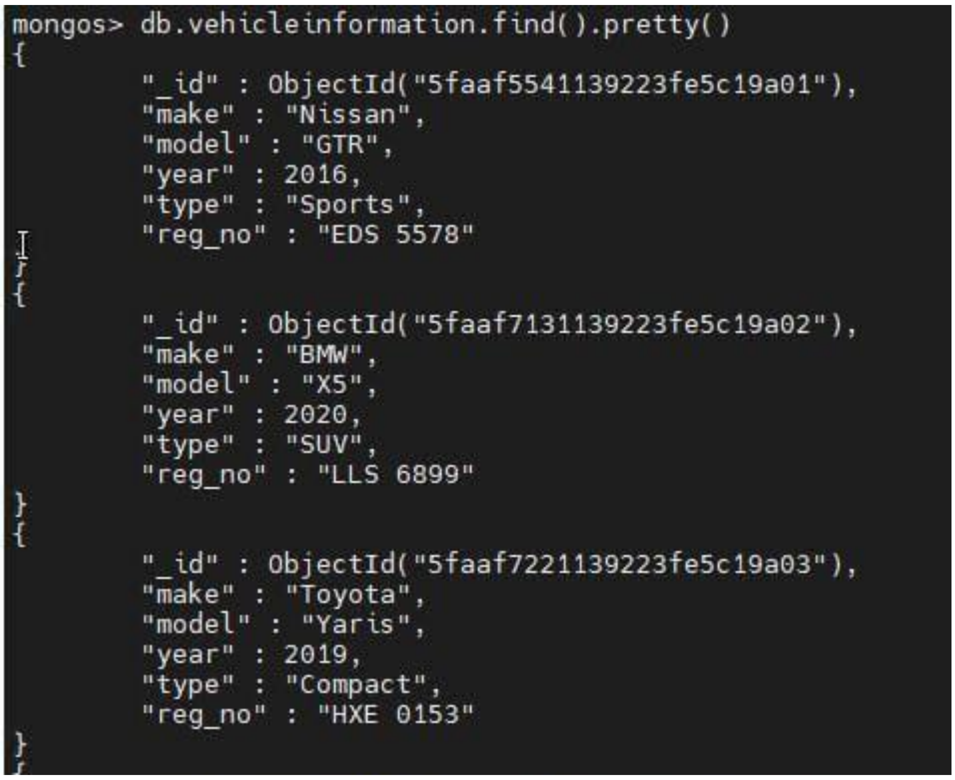
MongoDB projection uses the same find syntax, but we also add a set of parameters to the find function. This set of parameters informs the MongoDB instance of which fields to be returned.

Consider the following collection called "vehicle information"

Edit with WPS Office



```
{
        "_id" : ObjectId("5faaf7291139223fe5c19a04"),
        "make" : "Audi",
        "model" : "RS3",
        "year" : 2018,
        "type" : "Sports",
        "reg_no" : "RFD 7866"
}
{
        "_id" : ObjectId("5faaf72f1139223fe5c19a05"),
        "make" : "Ford",
        "model" : "Transit",
        "year" : 2017,
        "type" : "Van",
        "reg_no" : "TST 9880"
}
{
        "_id" : ObjectId("5faaf7371139223fe5c19a06"),
        "make" : "Honda",
        "model" : "Gold Wing",
        "year" : 2018,
        "type" : "Bike",
        "reg_no" : "LKS 2477"
}
mongos>
```



```
mongos> db.vehicleinformation.find().pretty()
{
        "_id" : ObjectId("5faaf5541139223fe5c19a01"),
        "make" : "Nissan",
        "model" : "GTR",
        "year" : 2016,
        "type" : "Sports",
        "reg_no" : "EDS 5578"
}
{
        "_id" : ObjectId("5faaf7131139223fe5c19a02"),
        "make" : "BMW",
        "model" : "X5",
        "year" : 2020,
        "type" : "SUV",
        "reg_no" : "LLS 6899"
}
{
        "_id" : ObjectId("5faaf7221139223fe5c19a03"),
        "make" : "Toyota",
        "model" : "Yaris",
        "year" : 2019,
        "type" : "Compact",
        "reg_no" : "HXE 0153"
}
```

Edit with WPS Office

GET SELECTED ATTRIBUTES

```
test> db.vehicle.find({},{model:1,year:1}).count();
```

IGNORED ATTRIBUTES

```
test>db.vehicle.find({},{_id:0}).count();
```

RETRIEVING SPECIFIC FIELDS FROM NESTED OBJECTS

$slice operator is used to retrieve specific field from nested objects.

```
db.findOne(
  {
    _id: "some_id",
    "array1._id": "level_1_id"
  },
  {
    fields: { _id: 0, "array1.$.array2": { $slice: [index, 1] }
  }
})
```

BENEFITS OF PROJECTION:

MongoDB projection is a powerful tool that can be used to extract only the fields you need from a document—not all fields. It enables you to:

Project concise and transparent data

Edit with WPS Office

Filter the data set without impacting the overall database performance.

LIMIT:

In MongoDB, the **limit()** method limits the number of records or documents that you want. It basically defines the max limit of records/documents that you want. Or in other words, this method uses on cursor to specify the maximum number of documents/ records the cursor will return. We can use this method after the find() method and find() will give you all the records or documents in the collection. You can also use some conditions inside the find to give you the result that you want.

In this method, we only pass numeric values.

This method is undefined for values which is less than -2^{31} and greater than 2^{31}.

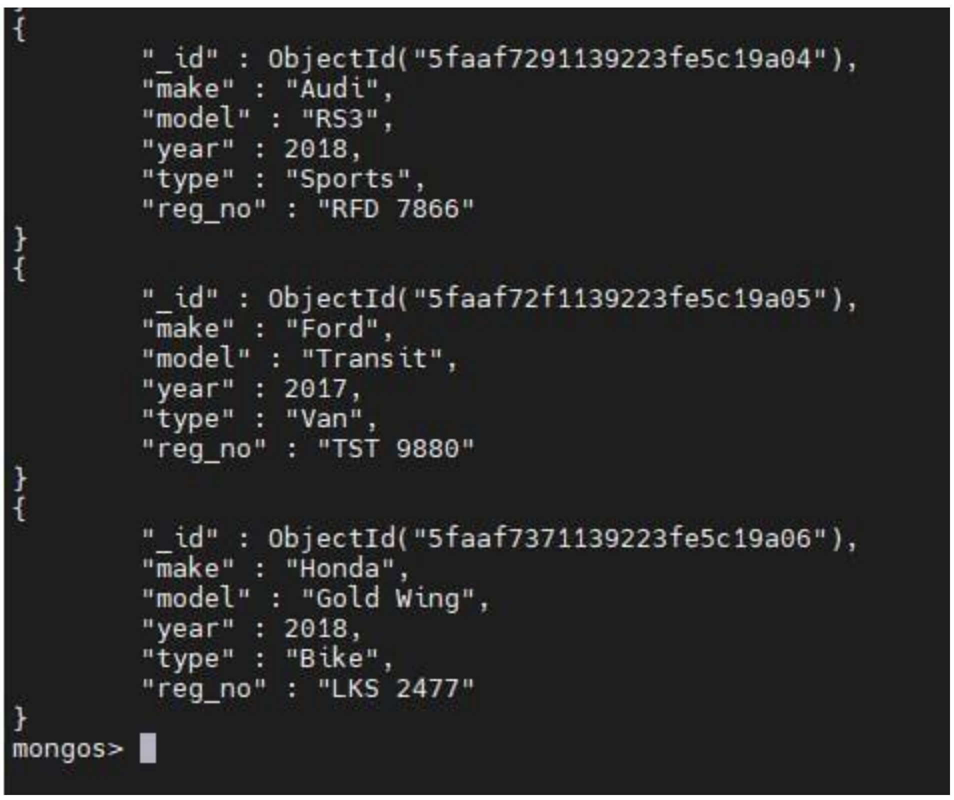
Passing 0 in this method(limit(0)) is equivalent to no limit.

Syntax:

cursor.limit()
$limit takes a number,n,and returns the first n resulting documents:

```
test>db.vehicle.find({},{_id:0}).limit(3);
```

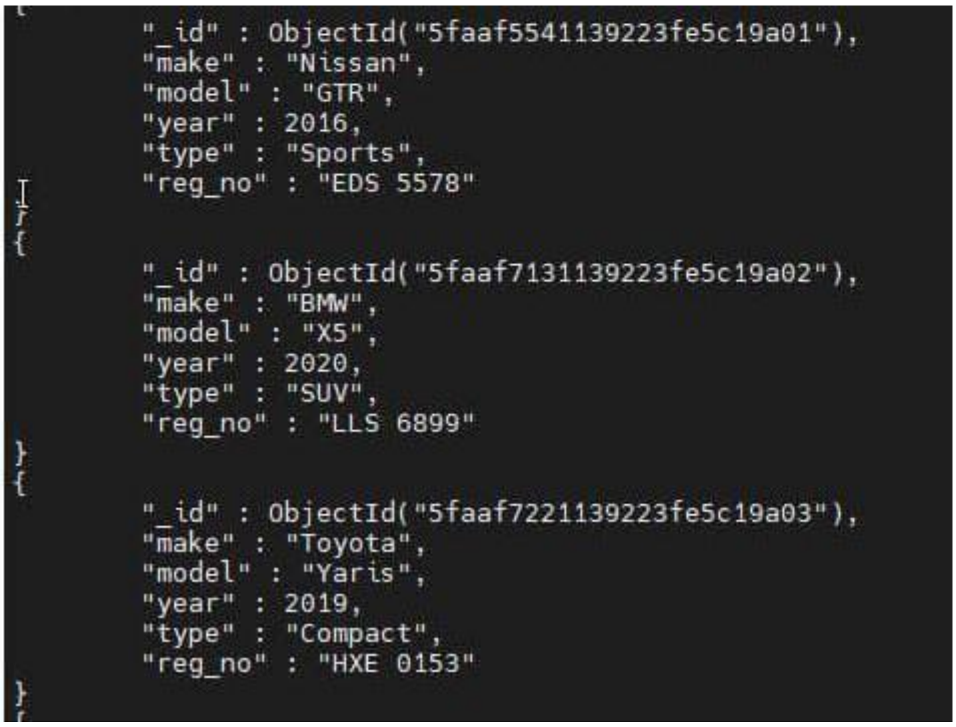
Edit with WPS Office



```
{
        "_id" : ObjectId("5faaf7291139223fe5c19a04"),
        "make" : "Audi",
        "model" : "RS3",
        "year" : 2018,
        "type" : "Sports",
        "reg_no" : "RFD 7866"
}
{
        "_id" : ObjectId("5faaf72f1139223fe5c19a05"),
        "make" : "Ford",
        "model" : "Transit",
        "year" : 2017,
        "type" : "Van",
        "reg_no" : "TST 9880"
}
{
        "_id" : ObjectId("5faaf7371139223fe5c19a06"),
        "make" : "Honda",
        "model" : "Gold Wing",
        "year" : 2018,
        "type" : "Bike",
        "reg_no" : "LKS 2477"
}
mongos> 
```

TOP 3 RESULTS:

Edit with WPS Office



```
        "_id" : ObjectId("5faaf5541139223fe5c19a01"),
        "make" : "Nissan",
        "model" : "GTR",
        "year" : 2016,
        "type" : "Sports",
        "reg_no" : "EDS 5578"
}
{
        "_id" : ObjectId("5faaf7131139223fe5c19a02"),
        "make" : "BMW",
        "model" : "X5",
        "year" : 2020,
        "type" : "SUV",
        "reg_no" : "LLS 6899"
}
{
        "_id" : ObjectId("5faaf7221139223fe5c19a03"),
        "make" : "Toyota",
        "model" : "Yaris",
        "year" : 2019,
        "type" : "Compact",
        "reg_no" : "HXE 0153"
}
```

SELECTORS:

COMPARISION:

$eq

Matches values that are equal to a specified value.

$gt

Matches values that are greater than a specified value.

$gte

Matches values that are greater than or equal to a specified value.

$in

Matches any of the values specified in an array.

$lt

Matches values that are less than a specified value.

$lte

Matches values that are less than or equal to a specified value.

$ne

Matches all values that are not equal to a specified value.

$nin

Matches none of the values specified in an array.

OR & AND OPERATOR:

Edit with WPS Office

OR: The$or operator performs a logical OR operation on an array of one or more <expressions> and selects the documents that satisfy at least one of the <expressions> .
Syntax:

```
{ $or: [ { <expression1> }, { <expression2> }, ... , { <expressionN> } ] }
```

AND: $and performs a logical AND operation on an array of one or more expressions (<expression1>, <expression2>, and so on) and selects the documents that satisfy all the expressions.

Syntax:

```
{ $and: [ { <expression1> }, { <expression2> } , ... , {
<expressionN> } ] }
```

BITWISE TYPES:

$bitsAllClear: Matches numeric or binary values in which a set of bit

$bitsAllSet:

positions all have a value of 0.

$bitsAnyClear

Matches numeric or binary values in which a set of bit positions all have a value of 1.

$bitsAnySet:

Matches numeric or binary values in which any bit from a set bit positions has a value of 0.

Matches numeric or binary values in which any bit from a set bit positions has a value of 1.

Edit with WPS Office

QUERY:

You can query documents in MongoDB by using the following methods:

Your programming language's driver.

1.The MongoDB Atlas UI To learn more, see Query Documents with MongoDB Atlas.

2.MongoDB Compass.

This page provides examples of query operations using the Collection.find() method in the MongoDB Node.js Driver.

The examples on this page use the inventory collection. Connect to a test database in your MongoDB instance then create the inventory collection:

```
await db.collection('inventory').insertMany([
 {
  item: 'journal',
  qty: 25,
  size: { h: 14, w: 21, uom: 'cm' },
  status: 'A'
 },
 {
  item: 'notebook',
  qty: 50,
```

Edit with WPS Office

```
  size: { h: 8.5, w: 11, uom: 'in' },

  status: 'A'

},

{

  item: 'paper',

  qty: 100,

  size: { h: 8.5, w: 11, uom: 'in' },

  status: 'D'

},

{

  item: 'planner',

  qty: 75,

  size: { h: 22.85, w: 30, uom: 'cm' },

  status: 'D'

},

{

  item: 'postcard',

  qty: 45,

  size: { h: 10, w: 15.25, uom: 'cm' },

  status: 'A'

}
```

Edit with WPS Office

```
]);
```

GEOSPATIAL:

MongoDB supports query operations on geospatial data. This section introduces MongoDB's geospatial features.

Compatibility

You can use geospatial queries for deployments hosted in the following environments:

MongoDB Atlas: The fully managed service for MongoDB deployments in the cloud

MongoDB Enterprise: The subscription-based, self-managed version of MongoDB

MongoDB Community: The source-available, free-to-use, and self-managed version of MongoDB

For deployments hosted in MongoDB Atlas, you can run geospatial queries in the UI by using the query **Filter** bar or aggregation builder. To learn more, see Perform Geospatial Queries in Atlas.

Geospatial Data

In MongoDB, you can store geospatial data as GeoJSON objects or as legacy coordinate pairs.

GeoJSON Objects

To calculate geometry over an Earth-like sphere, store your location data as GeoJSON objects.

Edit with WPS Office

To specify GeoJSON data, use an embedded document with:

a field named type that specifies the GeoJSON object type, and

a field named coordinates that specifies the object's coordinates.

```
<field>: { type: <GeoJSON type> , coordinates: <coordinates> }
```

If specifying latitude and longitude coordinates, list the **longitude** first, and then **latitude**.

Valid longitude values are between -180 and 180, both inclusive.

Valid latitude values are between -90 and 90, both inclusive.

For example, to specify a GeoJSON Point:

```
location: {
    type: "Point",
    coordinates: [-73.856077, 40.848447]
}
```

GEOSPATIAL QUERIES:

MongoDB provides the following geospatial query operators:

Name	Description
`$geoIntersects`	Selects geometries that intersect with a GeoJSON geometry. The 2dsphere index supports $geoIntersects.

Edit with WPS Office

Name	Description
$geoWithin	Selects geometries within a bounding GeoJSON geometry. The 2dsphere and 2d indexes support $geoWithin.
$near	Returns geospatial objects in proximity to a point. Requires a geospatial index. The 2dsphere and 2d indexes support $near.
$nearSphere	Returns geospatial objects in proximity to a point on a sphere. Requires a geospatial index. The 2dsphere and 2d indexes support $nearSphere.

DATATYPES AND OPERATIONS:

The datatypes in mongodb are

a.point: The following example specifies a GeoJSON Point:

{ type: "Point", coordinates: [40, 5] }

b.linestring: The following example specifies a GeoJSON LineString

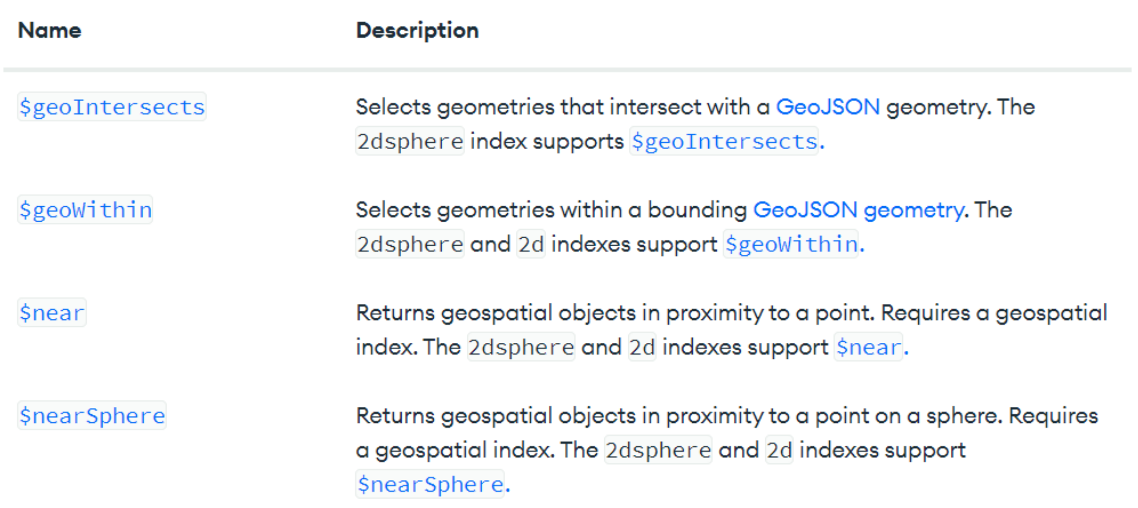
{ type: "LineString", coordinates: [[40, 5], [41, 6]] }

c.polygon:The following example specifies a GeoJSON Polygon:

```
{
  type: "Polygon",
  coordinates: [ [ [ 0 , 0 ] , [ 3 , 6 ] , [ 6 , 1 ] , [ 0 , 0  ] ] ]
}
```

Edit with WPS Office

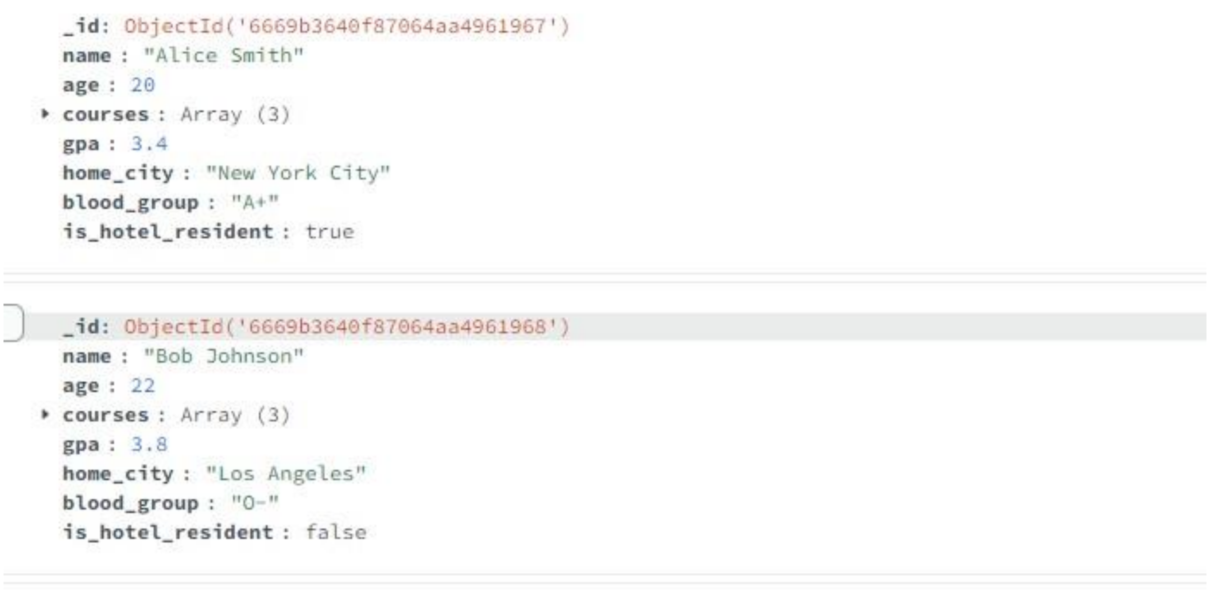
DATATYPES AND OPERATIONS:



Name	Description
`$geoIntersects`	Selects geometries that intersect with a GeoJSON geometry. The `2dsphere` index supports `$geoIntersects`.
`$geoWithin`	Selects geometries within a bounding GeoJSON geometry. The `2dsphere` and `2d` indexes support `$geoWithin`.
`$near`	Returns geospatial objects in proximity to a point. Requires a geospatial index. The `2dsphere` and `2d` indexes support `$near`.
`$nearSphere`	Returns geospatial objects in proximity to a point on a sphere. Requires a geospatial index. The `2dsphere` and `2d` indexes support `$nearSphere`.

Edit with WPS Office

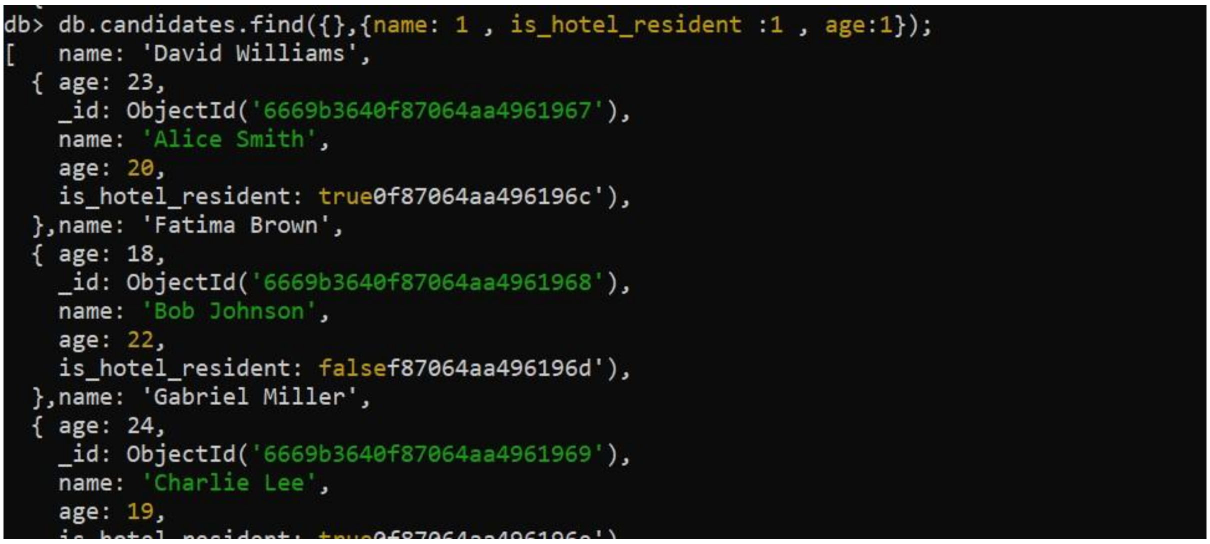
PROJECTION:



```
_id: ObjectId('6669b3640f87064aa4961967')
name : "Alice Smith"
age : 20
▸ courses : Array (3)
gpa : 3.4
home_city : "New York City"
blood_group : "A+"
is_hotel_resident : true
```

```
_id: ObjectId('6669b3640f87064aa4961968')
name : "Bob Johnson"
age : 22
▸ courses : Array (3)
gpa : 3.8
home_city : "Los Angeles"
blood_group : "O-"
is_hotel_resident : false
```

1.db.candidates.find({} , { name :1 , is_hotel_resident : 1 , age :1});



```
db> db.candidates.find({},{name: 1 , is_hotel_resident :1 , age:1});
[   name: 'David Williams',
  { age: 23,
    _id: ObjectId('6669b3640f87064aa4961967'),
    name: 'Alice Smith',
    age: 20,
    is_hotel_resident: true0f87064aa496196c'),
  },name: 'Fatima Brown',
  { age: 18,
    _id: ObjectId('6669b3640f87064aa4961968'),
    name: 'Bob Johnson',
    age: 22,
    is_hotel_resident: falsef87064aa496196d'),
  },name: 'Gabriel Miller',
  { age: 24,
    _id: ObjectId('6669b3640f87064aa4961969'),
    name: 'Charlie Lee',
    age: 19,
```

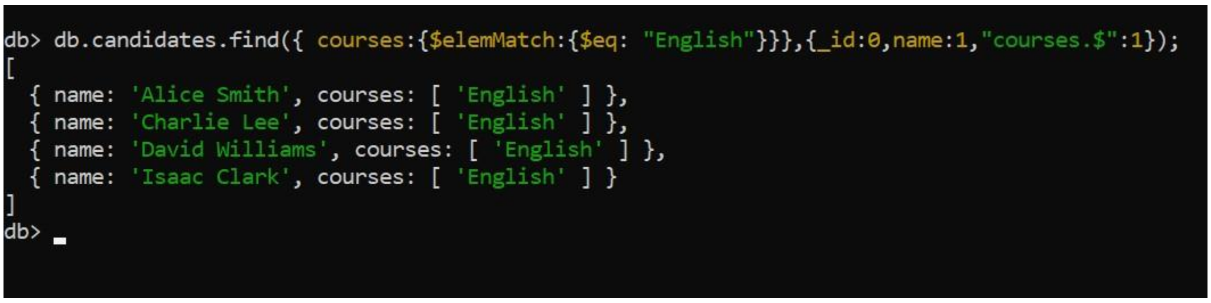
Edit with WPS Office

Projection Operator ($elemMatch) :

db.candidates.find({   courses : { $elemMatch:

{$eq : “English “}}} ,

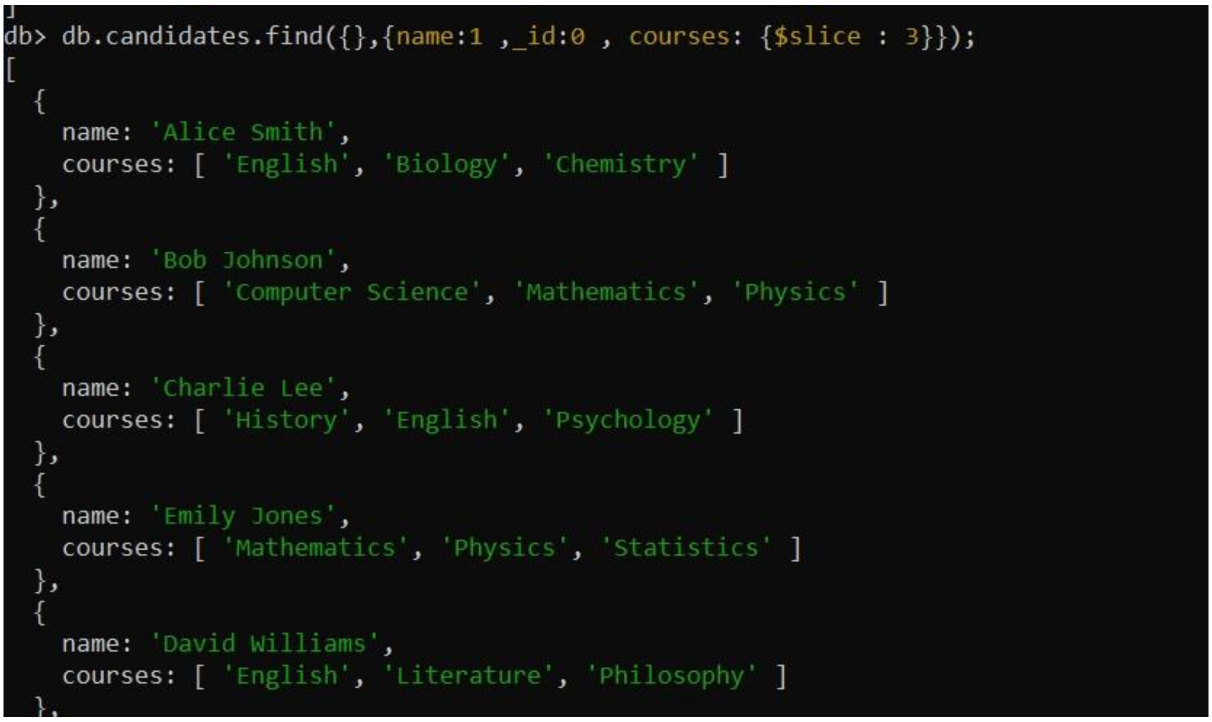
{ _id:0 , name:1 ,”courses.$” :1 }) ;



```
db> db.candidates.find({ courses:{$elemMatch:{$eq: "English"}}},{_id:0,name:1,"courses.$":1});
[
  { name: 'Alice Smith', courses: [ 'English' ] },
  { name: 'Charlie Lee', courses: [ 'English' ] },
  { name: 'David Williams', courses: [ 'English' ] },
  { name: 'Isaac Clark', courses: [ 'English' ] }
]
db> _
```

Projection Operator ($slice) :

db.candidates.find({ } ,{ name:1 , _id:0, courses :{ $slice:3}});



```
db> db.candidates.find({},{name:1 ,_id:0 , courses: {$slice : 3}});
[
  {
    name: 'Alice Smith',
    courses: [ 'English', 'Biology', 'Chemistry' ]
  },
  {
    name: 'Bob Johnson',
    courses: [ 'Computer Science', 'Mathematics', 'Physics' ]
  },
  {
    name: 'Charlie Lee',
    courses: [ 'History', 'English', 'Psychology' ]
  },
  {
    name: 'Emily Jones',
    courses: [ 'Mathematics', 'Physics', 'Statistics' ]
  },
  {
    name: 'David Williams',
    courses: [ 'English', 'Literature', 'Philosophy' ]
  },
```

Edit with WPS Office