

MONDO DB

CLASS 4: PROJECTION, LIMITS & SELECTORS

PROJECTIONS:

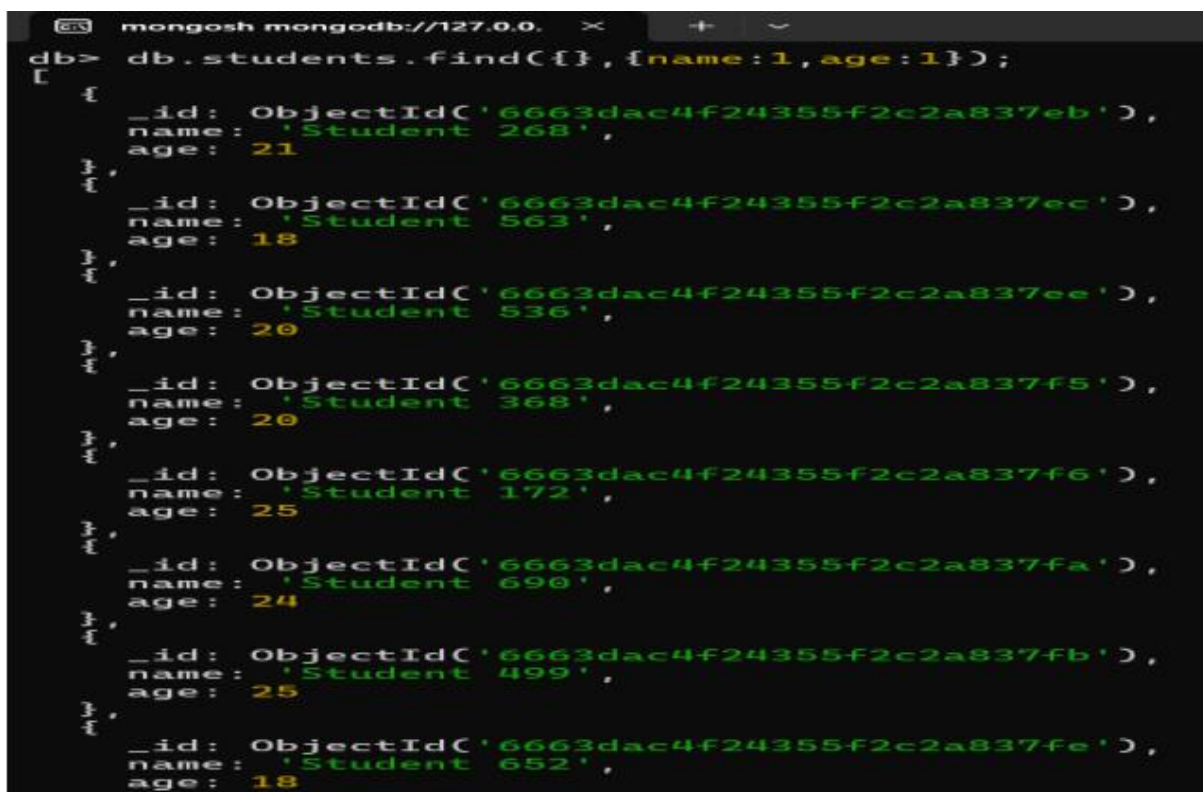
- Use the projection document as the second argument to the find method.
- Include field names with a value of 1 to specify fields to be returned.
- Omit fields or set them to 0 to exclude them from the results.

SELECTED ATTRIBUTES:

To filter a subset of attributes from a collection we use projection to get only a needed attributes.

```
db.students.find({}, {name:1, age:1, courses:1});
```

here we are trying to display only names and age of a students



```
mongosh mongod://127.0.0.1:27020
db> db.students.find({}, {name:1, age:1});
[
  {
    _id: ObjectId('6663dac4f24355f2c2a837eb'),
    name: 'Student 268',
    age: 21
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837ec'),
    name: 'Student 563',
    age: 18
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837ee'),
    name: 'Student 536',
    age: 20
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837f5'),
    name: 'Student 368',
    age: 20
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837f6'),
    name: 'Student 172',
    age: 25
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837fa'),
    name: 'Student 690',
    age: 24
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837fb'),
    name: 'Student 499',
    age: 25
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837fe'),
    name: 'Student 652',
    age: 18
  }
]
```

IGNORED ATTRIBUTES:

In MongoDB, there isn't a specific attribute or option called "ignored" under projection. However, you can achieve similar functionality by explicitly excluding fields you don't want in the query result using projection.

This attribute is used to print the exact data by excluding the object_id. Here _id is Used to find the exact student rather than searching here and there in the database It just saw the _id and find the specific group.

```

b> db.stud.find({}, {_id:0})
{
  name: 'Student 948',
  age: 19,
  courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
  gpa: 3.44,
  home_city: 'City 2',
  blood_group: 'O+',
  is_hotel_resident: true
},
{
  name: 'Student 157',
  age: 20,
  courses: "['Physics', 'English']",
  gpa: 2.27,
  home_city: 'City 4',
  blood_group: 'O-',
  is_hotel_resident: true
},
{
  name: 'Student 316',
  age: 20,
  courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
  gpa: 2.32,
  blood_group: 'B+',
  is_hotel_resident: true
},
{
  name: 'Student 346',
  age: 25,
  courses: "['Mathematics', 'History', 'English']",
  gpa: 3.31,
  home_city: 'City 8',

```

As mentioned above it removes all the _id of the collection and just give the Remaining instructions as there in the given collections.

RETRIVING SPECIFIC FIELDS FROM NESTED OBJECTS:

The \$slice operator in MongoDB is used to select a subset of an array. It is particularly useful when you have large arrays stored in your documents and you only need to retrieve certain elements optimizing data retrieval and reducing overhead.

This is used to get student name and only the mentioned number course from the course array.

```
db> db.students.find({}, { name: 1, courses: { $slice: 2 } });
[
  {
    _id: ObjectId('6663dac4f24355f2c2a837eb'),
    name: 'Student 268',
    courses: "['Mathematics', 'History', 'Physics']"
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837ec'),
    name: 'Student 563',
    courses: "['Mathematics', 'English']"
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837ee'),
    name: 'Student 536',
    courses: "['History', 'Physics', 'English', 'Mathematics']"
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837f5'),
    name: 'Student 368',
    courses: "['English', 'History', 'Physics', 'Computer Science']"
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837f6'),
    name: 'Student 172',
    courses: "['English', 'History', 'Physics', 'Mathematics']"
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837fa'),
    name: 'Student 690',
    courses: "['Computer Science', 'English', 'History']"
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837fb'),
    name: 'Student 499',
    courses: "['Mathematics', 'English', 'Computer Science', 'Physics']"
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837fe'),
```

BENEFITS OF PROJECTION:

- Efficiency
- Bandwidth Optimization
- Reduced Memory Usage
- Privacy and Security
- Improved Readability
- Optimized Index Usage

LIMITS:

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.

Applications of limit:

Performance Optimization: Limiting the number of documents retrieved can significantly improve query performance, especially when dealing with large collections.

Pagination: By combining limit with skipping techniques (e.g., using sort and skip), you can implement pagination in your application, efficiently retrieving specific pages of results.

Preventing Overwhelming Results: Limiting the number of documents returned helps avoid overwhelming your application or user interface with excessive data.

Syntax:

cursor.limit()

Or

db.collectionName.find(<query>).limit(<number>)

EXAMPLE:

To find only data of 5 students without an id

```
db.students.find({}, {_id:0}).limit(5);
```

```

db> db.students.find({}, {_id:0}).limit(5);
[
  {
    name: 'Student 268',
    age: 21,
    courses: "['Mathematics', 'History', 'Physics']",
    gpa: 4.48,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    name: 'Student 563',
    age: 18,
    courses: "['Mathematics', 'English']",
    gpa: 2.25,
    blood_group: 'AB+',
    is_hotel_resident: false
  },
  {
    name: 'Student 536',
    age: 20,
    courses: "['History', 'Physics', 'English', 'Mathematics']",
    gpa: 2.87,
    home_city: 'City 3',
    blood_group: 'O-',
    is_hotel_resident: false
  },
  {
    name: 'Student 368',
    age: 20,
    courses: "['English', 'History', 'Physics', 'Computer Science']",
    gpa: 4.41,
    home_city: 'City 9',
    blood_group: 'O-',
    is_hotel_resident: false
  },
  {
    name: 'Student 172',

```

LIMITING RESULTS:

It is used to limit query results in SQL . A MySQL supports the LIMIT clause to select a limited number of records. If we want to LIMIT the number of results that will return us a number of rows then we simply use the LIMIT command.

here to find only first five members data who have gpa greater than 3.7 without an id and courses

```
db.students.find({gpa:{>3.7}},{_id:0,courses:0}).limit(2);
```

```

db> db.students.find({}, {_id:0,courses:0}).limit(2);
[
  {
    name: 'Student 268',
    age: 21,
    gpa: 4.48,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    name: 'Student 563',
    age: 18,
    gpa: 2.25,
    blood_group: 'AB+',
    is_hotel_resident: false
  }
]
db> db.students.find({}, {_id:0,courses:0,_is_hotel_resident:0}).limit(2);
[
  {
    name: 'Student 268',
    age: 21,
    gpa: 4.48,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    name: 'Student 563',
    age: 18,
    gpa: 2.25,
    blood_group: 'AB+',
    is_hotel_resident: false
  }
]
db>

```

SELECTORS:

In MongoDB, **selectors** refer to query operators that allow you to filter and retrieve specific documents from a collection. These operators help you define conditions for querying data.

Here are some common uses of selectors:

Querying Data: You can use selectors to filter documents based on specific criteria.

Projection: Projection determines which fields to include or exclude in the query results. It's similar to the SELECT statement in SQL.

Limiting Results: The LIMIT equivalent in MongoDB limits the number of records returned by a query.

COMPARISON GT IT:

To find the students who are greater than 20 we use this command.

This is used to find the database greaterthan or lessthan

gt-greaterthan

lt-lessthan

Here the example to find all the students whose age is lessthan 30.

```
b> db.stud.find({age:{<20}});
{
  _id: ObjectId('665a89d776fc88153fffc09f'),
  name: 'Student 346',
  age: 25,
  courses: ["Mathematics", "History", "English"],
  gpa: 3.31,
  home_city: 'City 8',
  blood_group: 'O-',
  is_hotel_resident: true
},
{
  _id: ObjectId('665a89d776fc88153fffc0a0'),
  name: 'Student 930',
  age: 25,
  courses: ["English", "Computer Science", "Mathematics", "History"],
  gpa: 3.63,
  home_city: 'City 3',
  blood_group: 'A+',
  is_hotel_resident: true
},
{
  _id: ObjectId('665a89d776fc88153fffc0a1'),
  name: 'Student 305',
  age: 24,
  courses: ["History", "Physics", "Computer Science", "Mathematics"],
  gpa: 3.4,
  home_city: 'City 6',
  blood_group: 'O+',
  is_hotel_resident: true
}
```

Here the output will show the students whose age is lessthan 30.

AND OPERATOR:

The MongoDB \$and operator combines multiple conditions to form a logical “AND” relationship. It allows you to query for documents that meet **all** specified criteria.

Here is some Examples on AND operation

And operator is used to find the students who have gpa less than 3.8 and blood group “B+”

```
db> db.students.find({ $and: [{ gpa:{$lt:3.0} }, { blood_group: "B+" }] }).count();
16
db>
```

```
db> db.students.find({ $and: [{ gpa:{$lt:3.0} }, { blood_group: "B+" }] });
[
  {
    _id: ObjectId('6663dac4f24355f2c2a83817'),
    name: 'Student 610',
    age: 18,
    courses: "['Physics', 'History', 'Mathematics']",
    gpa: 2.58,
    home_city: 'City 5',
    blood_group: 'B+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a83834'),
    name: 'Student 367',
    age: 19,
    courses: "['English', 'Physics', 'History', 'Mathematics']",
    gpa: 2.81,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a8386d'),
    name: 'Student 252',
    age: 19,
    courses: "['History', 'Physics', 'Mathematics']",
    gpa: 2.8,
    blood_group: 'B+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a8387e'),
    name: 'Student 443',
    age: 22,
    courses: "['Computer Science', 'Mathematics']",
    gpa: 2.01,
    blood_group: 'B+',
    is_hotel_resident: false
  }
]
```

OR OPERATOR:

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>.

EXAMPLE: Here to find countings of students either having gpa greater than 3.4 and not a hotel resident.

```
type "it" for more
db> db.students.find({ $or: [{ is_hotel_resident:false},{gpa:{$gt:3.4}}]}).count();
236
```

```

db> db.students.find([{$or:[{is_hotel_resident:false},{gpa:{$gt:3.4}}]}]);
[
  {
    _id: ObjectId('6663dac4f24355f2c2a837eb'),
    name: 'Student 268',
    age: 21,
    courses: '["Mathematics", "History", "Physics"]',
    gpa: 4.48,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837ec'),
    name: 'Student 563',
    age: 18,
    courses: '["Mathematics", "English"]',
    gpa: 2.25,
    blood_group: 'AB+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837ee'),
    name: 'Student 536',
    age: 20,
    courses: '["History", "Physics", "English", "Mathematics"]',
    gpa: 2.87,
    home_city: 'City 3',
    blood_group: 'O-',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6663dac4f24355f2c2a837f5'),
    name: 'Student 368',
    age: 20,
    courses: '["English", "History", "Physics", "Computer Science"]',
    gpa: 4.41,
    home_city: 'City 9',
    blood_group: 'O-',
    is_hotel_resident: false
  },
]

```

BITWISE OPERATOR:

MongoDB provides **bitwise query operators** that allow you to filter data based on bit positions. Here are the available bitwise operators:

1. **\$bitsAllClear**: Matches documents where all specified bits are clear (i.e., set to 0).
2. **\$bitsAllSet**: Matches documents where all specified bits are set (i.e., set to 1).
3. **\$bitsAnyClear**: Matches documents where at least one of the specified bits is clear.
4. **\$bitsAnySet**: Matches documents where at least one of the specified bits is set.

You can use these operators in your queries to work with numeric or binary values.

{ field: { \$operator: value } }

↙ ↕ ↘

{ age: { \$lte: 30 } }

\$eq	Equal
\$ne	Not equal
\$gt	Greater than
\$gte	Greater than or equal to
\$lt	Less than
\$lte	Less than or equal to
\$in	In
\$nin	Not in

QUERY:

MongoDB Query is a fundamental aspect of MongoDB that allows users to fetch data from the database. Similar to **SQL** queries in traditional databases, MongoDB queries provide simplicity and flexibility in retrieving specific data based on certain criteria or conditions.

MongoDB Query

- [MongoDB Query](#) allows retrieving data from the MongoDB database. MongoDB Query provides simplicity in the process of fetching data from the [database](#), it's similar to SQL queries in [SQL](#) Database language.
- While performing a query operation, one can also use criteria or conditions that can be used to retrieve specific data from the database.
- MongoDB provides the function names as **db.collection_name.find()** to operate query operations on the database.

GEOSPATIAL QUERY:

A geospatial query involves retrieving information from a database based on geographic locations and spatial relationships. These queries are used in Geographic Information Systems (GIS) to analyze and visualize spatial data.

Need to upload a new collection called "location" in json format

Follow the same steps to switch this collection to database

Use db

Show dbs

Show collections

```
Current Mongosh Log ID: 66648228cfc68363b8cdcdf5
Connecting to:  mongodb://127.0.0.1:27017/?directConnection=true&
serverSelectionTimeoutMS=2000&appName=mongosh+2.2.6
Using MongoDB:  7.0.11
Using Mongosh:  2.2.6
```

For mongosh info see: <https://docs.mongodb.com/mongosh-shell/>

```
-----
The server generated these startup warnings when booting
2024-06-08T11:39:34.025+05:30: Access control is not enabled for the da
tabase. Read and write access to data and configuration is unrestricted
-----
```

```
test> use db
switched to db db
db> show dbs
admin  48.00 KiB
config 108.00 KiB
db      192.00 KiB
local   72.00 KiB
db> show collections
locations
students
students_permission
Please enter a MongoDB connection string (Default: mongodb://localhost/):
db> |
```

localhost:27017 > db > locations

Documents 5 Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [Gen](#) [Explain](#) [Reset](#) [Find](#) [Option](#)

[ADD DATA](#) [EXPORT DATA](#) 1-5 of 5 [≡](#) [⌵](#)

```
_id: 1
name: "Coffee Shop A"
location: Object
```

```
_id: 2
name: "Restaurant B"
location: Object
```

```
_id: 3
name: "Library C"
location: Object
```

```
_id: 4
name: "Museum D"
location: Object
```

Here to find a location.

```
db> db.locations.find({
... location:{
... $geoWithin:{
... $centerSphere:[[-74.005,40.712],0.00621376]]}}});
[
  {
    _id: 1,
    name: 'Coffee Shop A',
    location: { type: 'Point', coordinates: [ -73.985, 40.748 ] }
  },
  {
    _id: 2,
    name: 'Restaurant B',
    location: { type: 'Point', coordinates: [ -74.009, 40.712 ] }
  },
  {
    _id: 5,
    name: 'Park E',
    location: { type: 'Point', coordinates: [ -74.006, 40.705 ] }
  }
]
db>
```

GEOSPATIAL QUERY OPERATIONS:

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

MongoDB provides several geospatial query operators that allow you to work with geospatial data. Here are some of the key operators:

- **near:** This operator retrieves documents near a specified point. It looks up dataset points close to a given coordinates field.
- **\$geoWithin:** Use this operator to select points within a specified shape (e.g., a bounding GeoJSON shape).
- **\$geoIntersects:** This operator selects points that intersect a given geometry (supported by the 2dsphere index).
- **\$nearSphere:** specifies a point for which a geospatial query returns documents from nearest to farthest.

These operators enable powerful geospatial queries, allowing you to work with location-based data effectively.