PROJECTION , LIMIT AND SELECTORS

Projection:

Projection refers to the process of selecting specific fields to return from a query, effectively controlling which fields of the documents are include in the result set.

The projection work based on:

Indicates that the field should be include name to be returned.

Ex:{

“_id”:1,

“name”:”Alice”,

“age”:30,

“email”:”alice@example.com”,

“address”:”New York”,

“city”:”New York”,

“zipcode”:”10001”

}

}

Benefits of Projection:

- Reduces data transferred between the database and your application.
- Improves query performance by retrieving only necessary data.
- Simplifies your code by focusing on the specific information you need.

LIMIT:

- **Limit:**

- The `limit` operator is used with the `find` method.
- It's chained after the filter criteria or any sorting operations.

Syntax:

db.collection.find({filter},{projection}).limit(number)

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

```
db> db.students.find({},{_id:0}).limit(5);
[
  {
    name: 'Student 328',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English']",
    gpa: 3.42,
    home_city: 'City 2',
    blood_group: 'AB-',
    is_hotel_resident: true
  },
  {
    name: 'Student 468',
    age: 21,
    courses: "['Computer Science', 'Physics', 'Mathematics', 'History']",
    gpa: 3.97,
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    name: 'Student 504',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English', 'Mathematics']",
    gpa: 2.92,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    name: 'Student 915',
    age: 22,
    courses: "['Computer Science', 'History', 'Physics', 'English']",
    gpa: 3.37,
    blood_group: 'AB+',
    is_hotel_resident: true
  },
  {
    name: 'Student 367',
    age: 25,
    courses: "['History', 'Physics', 'Computer Science']",
    gpa: 3.11,
```

To get only first five documents we use limit(5).

Limiting Results:

- Limiting result in MongoDB is an important technique for controlling the amount of data returned by a query.
- This can help in optimizing performance, managing large datasets and implementing pagination.

Syntax: db. collection. find (<query criteria>).limit(<number>)

db. students. find ({gpa:{$gt:3.5}},{_id:0}).limit(2);

In this query this specifies that only documents where the gpa field is greater than 3.5 should be selected.

```
db> db.students.find({gpa:{$gt:3.5}},{_id:0}).limit(2);
[
  {
    name: 'Student 468',
    age: 21,
    courses: "['Computer Science', 'Physics', 'Mathematics', 'History']",
    gpa: 3.97,
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    name: 'Student 969',
    age: 24,
    courses: "['History', 'Mathematics', 'Physics', 'English']",
    gpa: 3.71,
    blood_group: 'B+',
    is_hotel_resident: true
  }
]
db>
```

If we want 10 top students result with condition id=0, and limit=5

db. students. find ({},{_id:0}).sort({_id:-1}).limit(5);

```
]
db> db.students.find({},{_id:0}).sort({_id:-1}).limit(5);
[
  {
    name: 'Student 933',
    age: 18,
    courses: "['Mathematics', 'English', 'Physics', 'History']",
    gpa: 3.04,
    home_city: 'City 10',
    blood_group: 'B-',
    is_hotel_resident: true
  },
  {
    name: 'Student 831',
    age: 20,
    courses: "['Mathematics', 'Computer Science']",
    gpa: 3.49,
    home_city: 'City 3',
    blood_group: 'AB+',
    is_hotel_resident: true
  },
  {
    name: 'Student 143',
    age: 21,
    courses: "['Mathematics', 'Computer Science', 'English', 'History']",
    gpa: 2.78,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    name: 'Student 718',
    age: 21,
    courses: "['Computer Science', 'English']",
    gpa: 2.75,
    home_city: 'City 5',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    name: 'Student 839',
```

This means the documents with the highest _id values will come first. with limit(5).

Selectors:

- Comparison gt and lt
- AND operator
- OR operator

Comparison gt and lt:

In MongoDB, comparison operators are used to query documents from a collection based on certain criteria. The "greater than" (`$gt`) and "less than" (`$lt`) operators are among these comparison operators.

Greater than: The $gt operator in MongoDB is used to specify a query condition where the field value must be greater than a specific value.it stands for “greater than.”

Lesser than: The less than operation is performed using the $lt operator. The `$lt` operator selects those documents where the value of the field is less than the specified value.

db. students. find ({age:{$gt:20}});

```
db> db.students.find({age:{$gt:20}});
[
  {
    _id: ObjectId('665609bde575a33e43e69479'),
    name: 'Student 328',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English']",
    gpa: 3.42,
    home_city: 'City 2',
    blood_group: 'AB-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665609bde575a33e43e6947c'),
    name: 'Student 468',
    age: 21,
    courses: "['Computer Science', 'Physics', 'Mathematics', 'History']",
    gpa: 3.97,
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665609bde575a33e43e6947d'),
    name: 'Student 504',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English', 'Mathematics']",
    gpa: 2.92,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665609bde575a33e43e6947f'),
    name: 'Student 915',
    age: 22,
    courses: "['Computer Science', 'History', 'Physics', 'English']",
    gpa: 3.37,
    blood_group: 'AB+',
    is_hotel_resident: true
  },
  {
```

This query gives the students collection with age is greater than 20.

AND operator:

To find students from “city 2” with blood group “B+”.

db. student. find ({$and :{{ home_city : ”City 2”}, { blood_group: ”B+”}});

```
Type "it" for more
db> db.students.find({$and:[{home_city:"City 2"},{blood_group:"B+"}]});
[
  {
    _id: ObjectId('665609bde575a33e43e6947d'),
    name: 'Student 504',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English', 'Mathematics']",
    gpa: 2.92,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665609bde575a33e43e6962b'),
    name: 'Student 872',
    age: 24,
    courses: "['English', 'Mathematics', 'History']",
    gpa: 3.36,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665df4511655576509ad81c8'),
    name: 'Student 504',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English', 'Mathematics']",
    gpa: 2.92,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665df4511655576509ad8376'),
    name: 'Student 872',
    age: 24,
    courses: "['English', 'Mathematics', 'History']",
    gpa: 3.36,
    home_city: 'City 2',
```

In this query is used to find the students data with using $and operator with home_city”2” and bloods group is “B+”.

OR operation:

db. student. find({$or:[{is_ hotel_ resident :true},{gpa:{$lt:3.0}}]});

The $or operator return documents that match at least one of the specified conditions.

```
]
db> db.students.find({$or:[{is_hotel_resident:true},{gpa:{$lt:3.0}}]});
[
  {
    _id: ObjectId('665609bde575a33e43e69479'),
    name: 'Student 328',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English']",
    gpa: 3.42,
    home_city: 'City 2',
    blood_group: 'AB-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665609bde575a33e43e6947c'),
    name: 'Student 468',
    age: 21,
    courses: "['Computer Science', 'Physics', 'Mathematics', 'History']",
    gpa: 3.97,
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665609bde575a33e43e6947d'),
    name: 'Student 504',
    age: 21,
    courses: "['Physics', 'Computer Science', 'English', 'Mathematics']",
    gpa: 2.92,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665609bde575a33e43e6947f'),
    name: 'Student 915',
    age: 22,
    courses: "['Computer Science', 'History', 'Physics', 'English']",
    gpa: 3.37,
    blood_group: 'AB+',
    is_hotel_resident: true
  },
```

In this query used to find the student data with using $or operator with hotel resident and gpa is less than 3.0.

Let’s Take new Data set:

- New Students Permission dataset link

Explanation: Collection name: students_permission

- **name:** Student's name (string)
- **age:** Student's age (number)
- **permissions:** Bitmask representing user permissions (number)

Bit wise:

bitwise query operators allow you to perform bitwise operations on numeric fields and match documents based on the result.

- In our example its a 32 bit each bit representing different things
- Bitwise value 7 means all access 7 -> 111

Bit 3	Bit 2	Bit 1
cafe	campus	lobby

Bit wise Types:

Bitwise operations in MongoDB allow you to perform operations on the individual bits of integer values stored in your documents. These operations are particularly useful for dealing with data that is naturally represented in binary form, such as flags or masks.

MongoDB supports several bitwise query operators that you can use to match documents based on the results of bitwise operations on field values. The main bitwise query operators in MongoDB are:

- **`$bitsAllClear`**: This operator matches documents where all the specified bits are clear (i.e., set to 0).

- **`$bitsAllSet`**: This operator matches documents where all the specified bits are set (i.e., set to 1).

- **`$bitsAnyClear`**: This operator matches documents where any of the specified bits are clear.

- **`$bitsAnySet`**: This operator matches documents where any of the specified bits are set.

Example:

- **`$bitsAllClear`**: Find documents where all the specified bits are clear.

```
db. Inventory.find({ flags: { $bitsAllClear: 6 } })
```

  - This query finds documents where both the 2nd (binary 10) and 3rd (binary 100) bits are 0 in the `flags` field.

- **`$bitsAllSet`**: Find documents where all the specified bits are set.

```
db.inventory.find({ flags: { $bitsAllSet: 5 } })
```

  - This query finds documents where both the 1st (binary 1) and 3rd (binary 100) bits are 1 in the `flags` field.

- **`$bitsAnyClear`**: Find documents where any of the specified bits are clear.

```
db.inventory.find({ flags: { $bitsAnyClear: 9 } })
```

  - This query finds documents where either the 1st (binary 1) or 4th (binary 1000) bits are 0 in the `flags` field.

- **`$bitsAnySet`**: Find documents where any of the specified bits are set.

```
db.inventory.find({ flags: { $bitsAnySet: 12 } })
```

  - This query finds documents where either the 3rd (binary 100) or 4th (binary 1000) bits are 1 in the `flags` field.

Query:

MongoDB queries using bitwise operators, you need to understand how these operators work and how to structure your queries properly.

Ex: const LOBBY_PERMISSION=1;

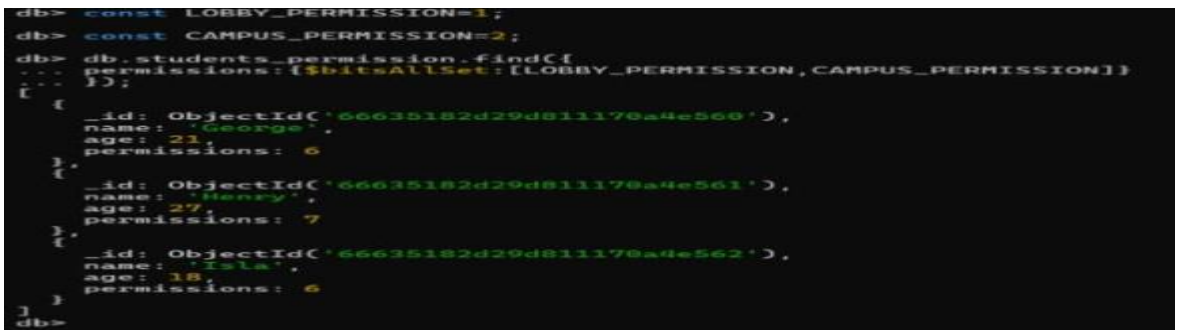
const CAMPUS_PERMISSION =2;

db .students_ permission. find({permissions:{

$bitsAllSet:

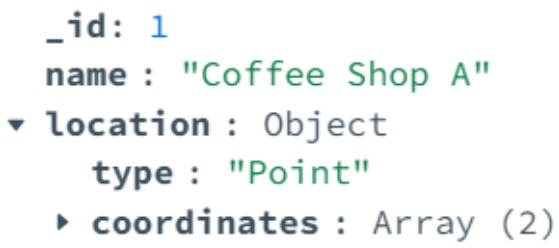
[LOBBY_PERMISSION,CAMPUS_PERMISSION]}});

Output:



Geosptial:

- Official Documentation link
- Create collection called “locations”
- Upload the dataset using json link



Geospatial Query:

Geospatial queries in MongoDB leverage geospatial indexes to efficiently execute spatial queries. MongoDB supports two types of geospatial indexes: `2d` indexes for planar (Euclidean) data and `2dsphere` indexes for spherical data.

db. location. find({

location:{

$geoWithin:{

$centerSphere:[[-74.005,40.712],0.00621376]}

}});

```
db> db.locations.find({
...   location: {
...     $geoWithin: {
...       $centerSphere: [[-74.005, 40.712], 0.00621376] // 1 kilometer in radians
...     }
...   }
... });
[
  {
    _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>
```

Data types and Operations:

Data Type:

- ✓ point
- ✓ Line String
- ✓ Polygon

Data types 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`.