Session 4

Projection, Limit & Selectors

-
- 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.

Get Selected Attributes

- Given a Collection you want to FILTER a subset of attributes. That is the place Projection is used.

```
// Get only the name and age for all students
db.students.find({}, { name: 1, age: 1 });
```

Ignore Attributes

```
// Get all student data but exclude the _id field
db.students.find({}, { _id: 0 });
```

Retrieving Specific Fields from Nested Objects

```
// Get student name and only the first course from the courses array
db.students.find({}, {
  name: 1,
  courses: { $slice: 1 }
});
```

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

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)

Get First 5 document

```
// Assuming you have already executed a query on the student collection
// Limit the results to the first 5 documents
db.students.find({}, { _id: 0 }).limit(5);
```

Limiting Results

```
// Find all students with GPA greater than 3.5 and limit to 2 documents
db.students.find({ gpa: { $gt: 3.5 } }, { _id: 0 }).limit(2);
```

I want Top 10 Results

```
// Sort documents in descending order by _id and limit to 5
db.students.find({}, { _id: 0 }).sort({ _id: -1 }).limit(5);
```

Experiment 3 - Selectors

Comparison gt lt

```
// Find all students with age greater than 20
db.students.find({ age: { $gt: 20 } });
```

AND operator

```
// Find students from "City 2" with blood group "B+"
db.students.find({
  $and: [
    { home_city: "City 2" },
    { blood_group: "B+" }
  ]
});
```

OR operator

```
// Find students who are hotel residents OR have a GPA less than 3.0
db.students.find({
  $or: [
    { is_hotel_resident: true },
    { gpa: { $lt: 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)

Bitwise Value

- 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

Bitwise Types

Bitwise Types

Bitwise

Name	Description
`$bitsAllClear`	Matches numeric or binary values in which a set of bit positions *all* have a value of `0`.
`$bitsAllSet`	Matches numeric or binary values in which a set of bit positions *all* have a value of `1`.
`$bitsAnyClear`	Matches numeric or binary values in which *any* bit from a set of bit positions has a value of `0`.
`$bitsAnySet`	Matches numeric or binary values in which *any* bit from a set of bit positions has a value of `1`.

Query

```
// Define bit positions for permissions
const LOBBY_PERMISSION = 1;
const CAMPUS_PERMISSION = 2;

// Query to find students with both lobby and campus permissions using
db.students_permission.find({
  permissions: { $bitsAllSet: [LOBBY_PERMISSION, CAMPUS_PERMISSION] }
});
```

Geospatial

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

```
_id: 1
name : "Coffee Shop A"
location : Object
  type : "Point"
  coordinates : Array (2)
```

Geospatial Query

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

Output

```
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

DataType

- 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`.