MONGODB

CLASS 4: PROJECTION , LIMIT & SELECTORS

PROJECTION

In MongoDB, projections are used to specify which fields you want to retrieve from a document when querying your database. This allows you to retrieve only the relevant data you need, instead of fetching the entire document, which can improve performance and bandwidth usage.

Specifying Fields:

- You can include or exclude fields using a projection document passed as the second argument to the `find` method.
- To include a field, set its value to `1` or `true`.
- To exclude a field (including the default `_id` field), set its value to `0` or `false`.

Example:

Let's say you have a collection named `products` with documents containing fields like `name`, `price`, `description`, and `category`. You only need the `name` and `price` for your current task. Here's how to achieve this using projection:

db.products.find({}, { name: 1, price: 1 })

This query will find all products (empty first argument `{}`) and return only the `name` and `price` fields for each product due to the projection document `{ name: 1, price: 1 }`.

BENEFITS

By only returning the fields you need, projections significantly reduce the amount of data that needs to be transferred between the database server and your application. This can lead to faster query execution times, especially when dealing with large documents or remote connections.

LIMIT

The `LIMIT` operator is used in MongoDB to restrict the number of documents returned by a query. It's a fundamental tool for efficient data retrieval, especially when dealing with large collections.

Syntax:

db.collection.find({ query }, { limit: <positive_integer> })

Example:

db.products.find({}, { limit: 10 })

This query retrieves the first 10 documents (sorted by the natural order of the `_id` field) from the `products` collection, regardless of any other criteria.

FIELD PROJECTION

Field projection allows you to specify which fields you want to retrieve from a document when querying your database. This is a powerful technique for optimizing performance, bandwidth usage, and data security

GET FIRST 5 DOCUMENTS

```
db> db.students.find({},{_id:0}).limit(5);
[
  {
    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: 2.77,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.82,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 3.31,
    home_city: 'City 8',
    blood_group: 'O-',
    is_hotel_resident: true
  },
```

The query 'db.students.find({}, {_id:0}).limit(5);' is used to retrieve documents from the 'students' collection in MongoDB. The query returns up to five student documents but excludes the_id' field in the results. The output includes details such as the student's name, age, courses, GPA, home city, blood group, and whether they are a hotel resident. This demonstrates the use of the 'find method with a projection to exclude specific fields and the 'limit' method to restrict the number of documents returned.

LIMITING RESULTS

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

The query 'db.students.find({gpa: {$gt:3.5}},{_id:0}).limit(2); retrieves documents from the 'students' collection in MongoDB where the 'gpa' field is greater than 3.5. It excludes the _id' field from the results and limits the output to two documents. The resulting documents include details such as the student's name, age, courses, GPA, home city, blood group, and whether they are a hotel resident, showcasing the use of the 'find method with filtering criteria, projection, and limit.

TO GET TOP 10 RESULTS

```
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
  },
```

The query 'db.students.find({}, {_id:0}).sort({_id:-1}).limit(5); retrieves documents from the 'students' collection, excluding the '_id' field from the results, and sorts them in descending order by the '_id' field. The 'sort({_id:-1})' part ensures that the documents are ordered from the most recent to the oldest based on their '_id' values. The 'limit(5)' clause restricts the output to the first five documents that meet these criteria. The resulting documents include various student details such as name, age, courses, GPA, home city, blood group, and whether they are hotel residents, demonstrating how sorting and limiting can be applied to refine query results

SELECTORS

In MongoDB, selectors are the core component for specifying which documents you want to retrieve from a collection. They act as filters that narrow down the results based on certain criteria.

Comparison gt lt

```
db> db.students.find({age:{$gt:20}});
[
  {
    _id: ObjectId('6649bb89b51b15a423b44ad0'),
    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('6649bb89b51b15a423b44ad1'),
    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('6649bb89b51b15a423b44ad2'),
    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
  },
  {
    _id: ObjectId('6649bb89b51b15a423b44ad5'),
    name: 'Student 440',
    age: 21,
    courses: "['History', 'Physics', 'Computer Science']",
    gpa: 2.56,
    home_city: 'City 10',
    blood_group: 'O-',
    is_hotel_resident: true
```

In MongoDB, `$gt` and `$lt` are comparison operators used for filtering documents based on field values. They stand for "greater than" and "less than," respectively. Here's a detailed explanation of their usage:

$gt (Greater Than):

- Selects documents where the value of a specified field is greater than a particular value.
- Syntax: `{ field_name: { $gt: value } }`
- Example: `{ age: { $gt: 25 } }` (finds documents where the `age` field is greater than 25)

$lt (Less Than):

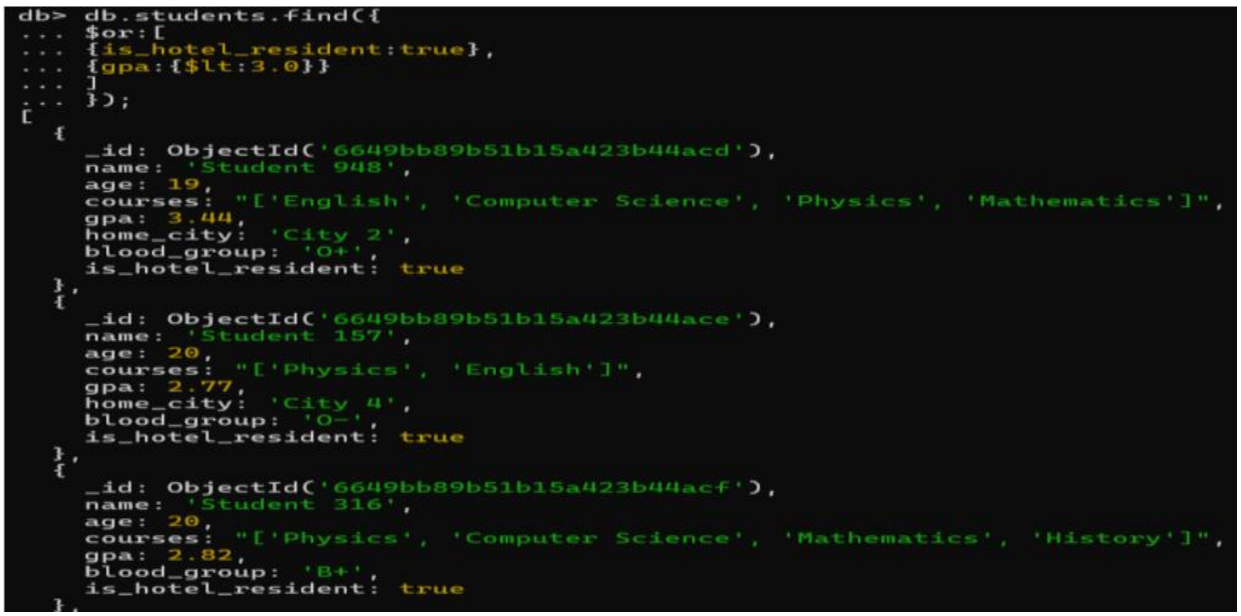
- Selects documents where the value of a specified field is less than a particular value.
- Syntax: `{ field_name: { $lt: value } }`
- Example: `{ price: { $lt: 100 } }` (finds documents where the `price` field is less than 100)

AND operator

```
db> db.students.find({
... $and:[
... {home_city:"City 2"},
... {blood_group:"B+"}
... ]
... });
[
  {
    _id: ObjectId('6649bb89b51b15a423b44ae5'),
    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('6649bb89b51b15a423b44c93'),
    name: 'Student 872',
    age: 24,
    courses: "['English', 'Mathematics', 'History']",
    gpa: 3.36,
    home_city: 'City 2',
    blood_group: 'B+',
    is_hotel_resident: true
  }
]
db>
```

The `$and` operator in MongoDB is a logical operator used to combine multiple conditions within a query. It allows you to specify that a document must satisfy **all** of the conditions provided in the `$and` array to be included in the results.

OR operator



The `$or` operator in MongoDB is another essential logical operator used in queries. It allows you to specify that a document must satisfy **at least one** of the conditions provided in the `$or` array to be included in the results.

Let’s Take new Data set

Here's an example of inserting a new dataset into MongoDB using the `mongoimport` command-line tool:

Scenario:

Imagine you have a CSV file named `books.csv` containing information about books, including:

- `title` (string)
- `author` (string)
- `year` (integer)
- `genre` (string)

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

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

In MongoDB, queries are the fundamental way to retrieve data from your collections. They allow you to filter and select specific documents based on various criteria.

```
db> const LOBBY_PERMISSION=1;

db> const CAMPUS_PERMISSION=2;

db> db.students_permission.find({
... permissions:{$bitsAllSet:[LOBBY_PERMISSION,CAMPUS_PERMISSION]}
... });
[
  {
    _id: ObjectId('66635182d29d811170a4e560'),
    name: 'George',
    age: 21,
    permissions: 6
  },
  {
    _id: ObjectId('66635182d29d811170a4e561'),
    name: 'Henry',
    age: 27,
    permissions: 7
  },
  {
    _id: ObjectId('66635182d29d811170a4e562'),
    name: 'Isla',
    age: 18,
    permissions: 6
  }
]
db>
```

The MongoDB query utilizes the 'find' method with the 'SbitsAllSet operator to retrieve documents from the 'students_permission' collection where specific bits in the permissions' field are set. The 'SbitsAllSet operator checks if all specified bit positions are set to 1 in the binary representation of the 'permissions` field.

In this query, the bit positions represented by 'LOBBY_PERMISSION and *CAMPUS_PERMISSION are checked. These constants represent specific permission bits in a binary format. The query returns documents where both of these bits are set in the 'permissions' field.

Geospatial

In MongoDB, geospatial queries allow you to search for data based on geographic location. This functionality is incredibly useful for applications that deal with maps, locations, or nearby searches.

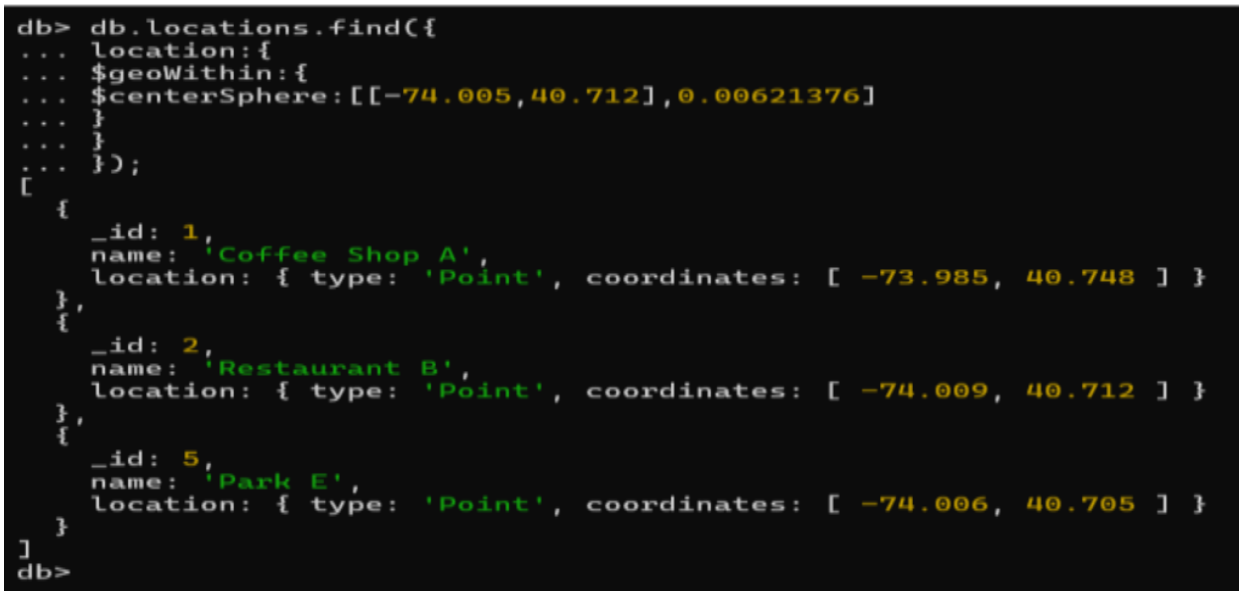
Example:

Assuming you have a collection named `restaurants` with a GeoJSON field named `location` storing restaurant coordinates, here's how to find restaurants near a specific latitude and longitude:

```
// Find restaurants within 1 kilometer of [50.0646501, 19.9417592] (Krakow, Poland)
db.restaurants.find({
  location: {
    $nearSphere: {
      $geometry: { type: "Point", coordinates: [50.0646501, 19.9417592] },
      $maxDistance: 1000 // Maximum distance in meters (1 kilometer)
    }
  }
})
```

Geospatial Query

In MongoDB, geospatial queries enable you to search for data based on geographic location. This is essential for applications dealing with maps, finding nearby locations, or implementing location-based features.



The given MongoDB query is using the $geoWithin operator with $centerSphere to find locations within a specified radius from a central point. In this case, it's searching for locations within approximately 500 meters (0.00621376 radians) from the coordinates [-74.005, 40.712], which is a point in New York City. The result includes three locations: Coffee Shop A, Restaurant B, and Park E, along with their respective coordinates.

Data types and Operations

In MongoDB, the data types include String, Number, Boolean, Object, Array, Null, and Date, among others.

Point: A point is a GeoJSON object representing a single geographic coordinate.

LineString: A LineString is a GeoJSON object representing a sequence of connected line segments.

Polygon: A Polygon is a GeoJSON object representing a closed, two-dimensional shape with three or more vertices

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`.