MongoDB

PROJECTION , LIMIT & SELECTORS:

PROJECTION:

Projections are a powerful tool in MongoDB that allow you to control which fields are returned from your queries.

Understanding Projection Syntax:

The projection document is used within the `find` method to specify which fields to include or exclude in the returned documents.

Here's the basic syntax:

db.collection.find({ filter }, { projection: { field1: value, field2: value } })

- `filter`: This is an optional document that defines which documents to match in the collection (similar to selectors in a WHERE clause).
- `projection`: This is a document that specifies which fields to include or exclude from the returned documents.
- `field1`: The name of the field you want to include or exclude.
- `value`:
    - `1`: Include the specified field.
    - `0`: Exclude the specified field.

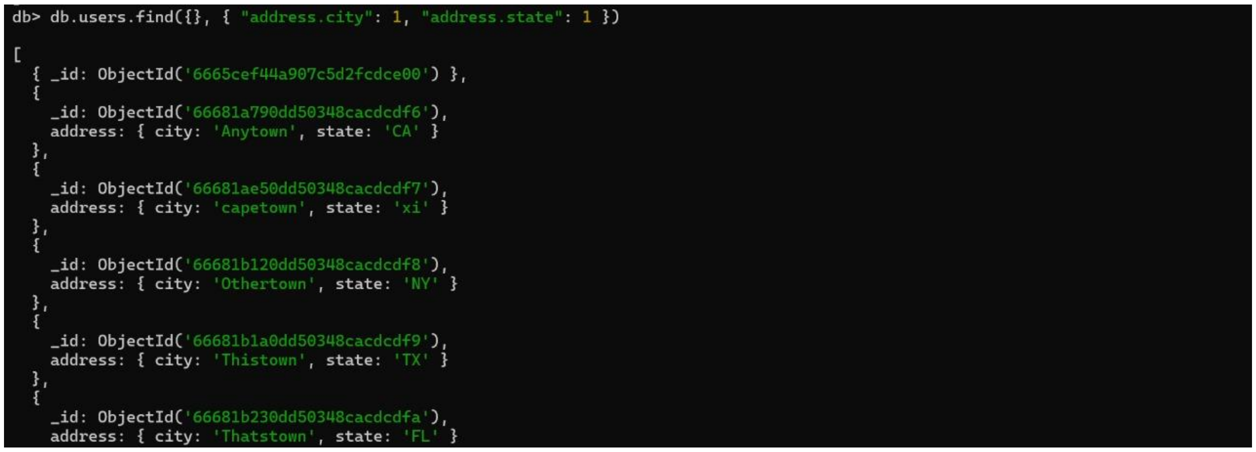
Specifying Field Inclusion:

To include specific fields in the query results, set their corresponding values to 1 in the projection document:

```
// Retrieve only name and price from products, excluding _id
db.products.find({}, { projection: { _id: 1, name: 0, price: 1 } });
```

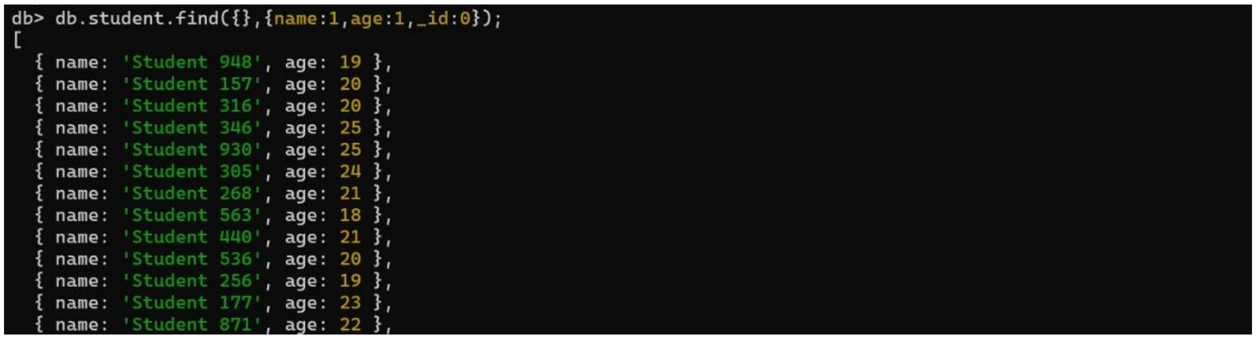
In this example, the _id field is excluded (set to 0), while name and price are included (set to 1).

With _id:



```
db> db.users.find({}, { "address.city": 1, "address.state": 1 })
[
  { _id: ObjectId('6665cef44a907c5d2fcdce00') },
  {
    _id: ObjectId('66681a790dd50348cacdcdf6'),
    address: { city: 'Anytown', state: 'CA' }
  },
  {
    _id: ObjectId('66681ae50dd50348cacdcdf7'),
    address: { city: 'capetown', state: 'xi' }
  },
  {
    _id: ObjectId('66681b120dd50348cacdcdf8'),
    address: { city: 'Othertown', state: 'NY' }
  },
  {
    _id: ObjectId('66681b1a0dd50348cacdcdf9'),
    address: { city: 'Thistown', state: 'TX' }
  },
  {
    _id: ObjectId('66681b230dd50348cacdcdfa'),
    address: { city: 'Thatstown', state: 'FL' }
```

Without _id:



```
db> db.student.find({},{name:1,age:1,_id:0});
[
  { name: 'Student 948', age: 19 },
  { name: 'Student 157', age: 20 },
  { name: 'Student 316', age: 20 },
  { name: 'Student 346', age: 25 },
  { name: 'Student 930', age: 25 },
  { name: 'Student 305', age: 24 },
  { name: 'Student 268', age: 21 },
  { name: 'Student 563', age: 18 },
  { name: 'Student 440', age: 21 },
  { name: 'Student 536', age: 20 },
  { name: 'Student 256', age: 19 },
  { name: 'Student 177', age: 23 },
  { name: 'Student 871', age: 22 },
```

Specifying Field Exclusion:

You can also exclude unwanted fields by setting their values to 0:

```
// Retrieve all user fields except email

db.users.find({}, { projection: { email: 0 } });
```

Here, all user fields are returned except for the email field, which is excluded.

Default Projection Behavior:

- If you don't specify a projection, MongoDB returns all fields by default, including the _id field.

- Specifying a field for inclusion implicitly excludes all other fields unless you explicitly include _id: 1.

Here, all user fields are returned except for the email field, which is excluded.

Default Projection Behavior:

- If you don't specify a projection, MongoDB returns all fields by default, including the _id field.
- Specifying a field for inclusion implicitly excludes all other fields unless you explicitly include _id: 1.

Advanced Projection Techniques:

- **Nested Projections:** You can use nested projections to control field inclusion within embedded documents.
- **Projection Operators:** MongoDB offers operators like $slice and $elemMatch for more granular control over projected data from arrays.

LIMIT:

Controlling the Number of Retrieved Documents

The `limit` operator in MongoDB allows you to restrict the number of documents returned by a query. This functionality is crucial for efficient data retrieval, especially when dealing with large collections.

Understanding Limit Syntax:

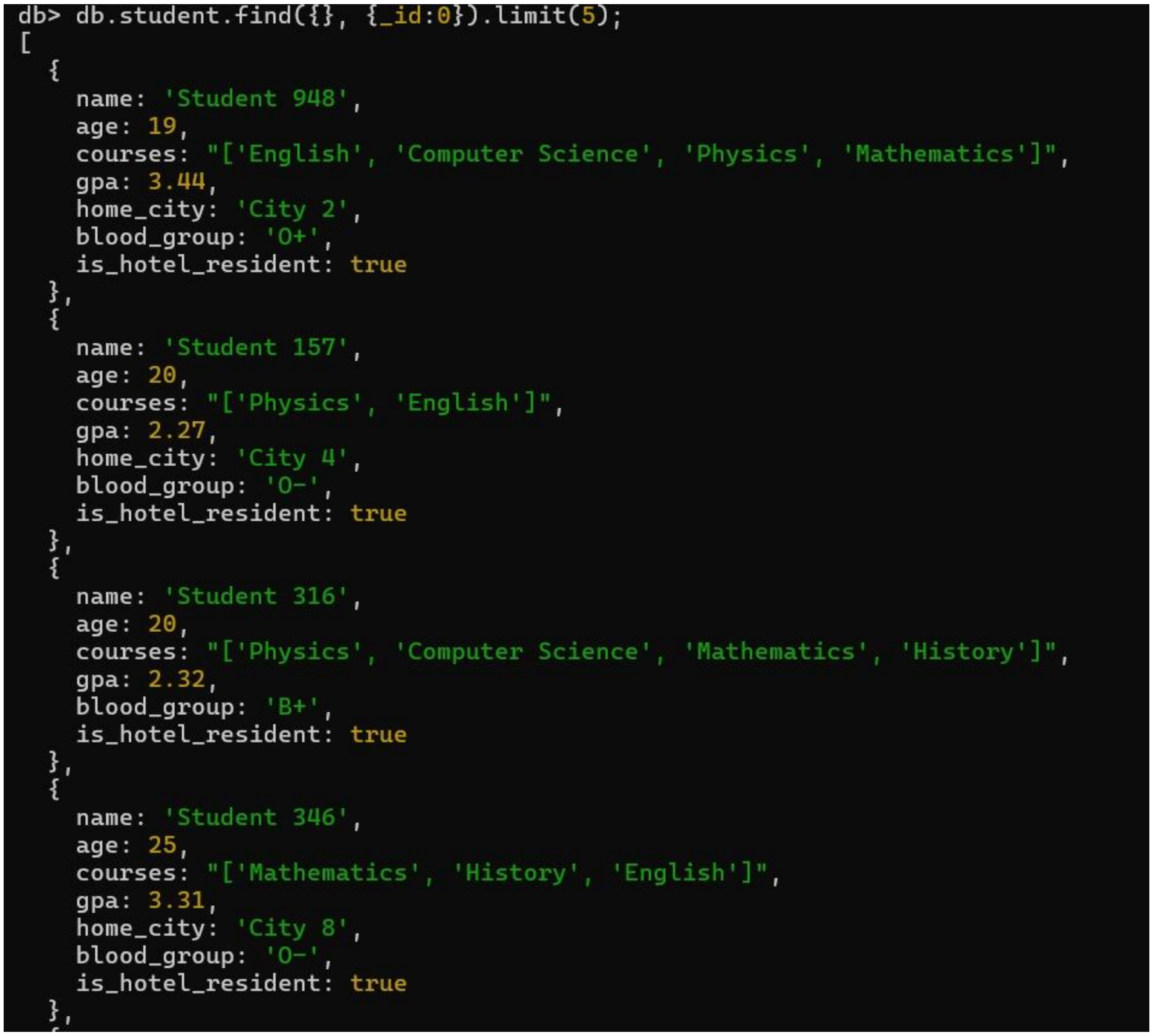
The `limit` operator is used within the `find` method to specify the maximum number of documents you want to retrieve.

Here's the basic syntax:

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

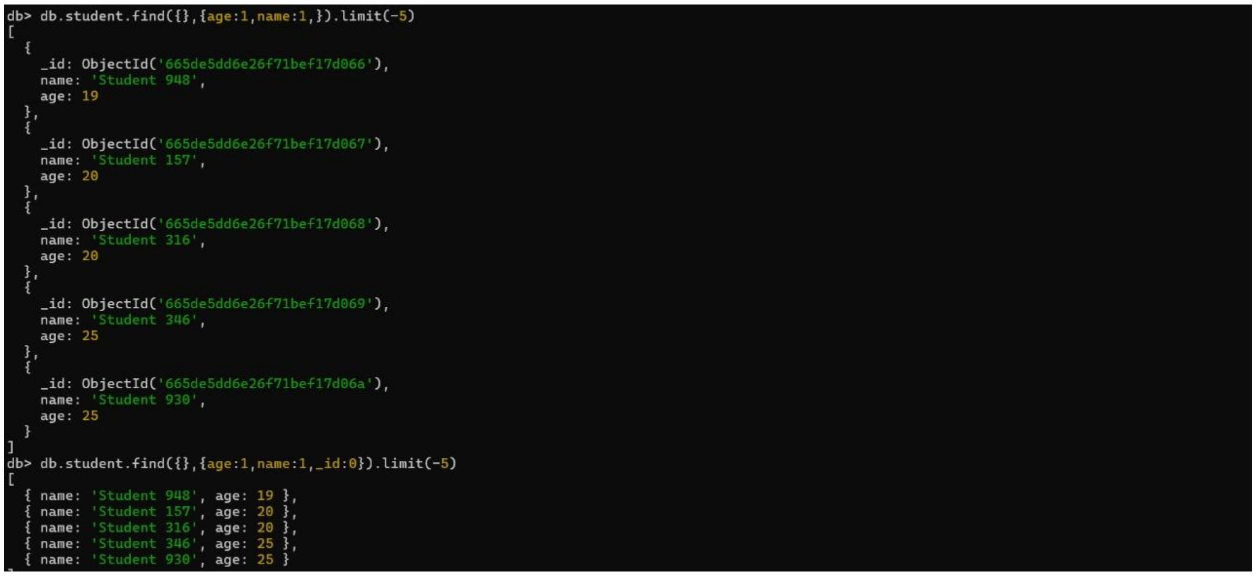
- **`filter`**: This is an optional document that defines which documents to match in the collection (similar to selectors in a WHERE clause).

- **`limit`:** This is a numeric value representing the maximum number of documents to return from the query.
  Example:
  // Retrieve the first 10 users sorted by name (ascending)
  db.users.find({}, { limit: 10, sort: { name: 1 } });
  In this example, the query retrieves a maximum of 10 users sorted alphabetically by their name field in ascending order (1 for ascending, -1 for descending).



```
db> db.student.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: 2.27,
    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',
    blood_group: 'O-',
    is_hotel_resident: true
  },
```

NEGATIVE   IINDEX:



```
db> db.student.find({},{age:1,name:1,}).limit(-5)
[
  {
    _id: ObjectId('665de5dd6e26f71bef17d066'),
    name: 'Student 948',
    age: 19
  },
  {
    _id: ObjectId('665de5dd6e26f71bef17d067'),
    name: 'Student 157',
    age: 20
  },
  {
    _id: ObjectId('665de5dd6e26f71bef17d068'),
    name: 'Student 316',
    age: 20
  },
  {
    _id: ObjectId('665de5dd6e26f71bef17d069'),
    name: 'Student 346',
    age: 25
  },
  {
    _id: ObjectId('665de5dd6e26f71bef17d06a'),
    name: 'Student 930',
    age: 25
  }
]
db> db.student.find({},{age:1,name:1,_id:0}).limit(-5)
[
  { name: 'Student 948', age: 19 },
  { name: 'Student 157', age: 20 },
  { name: 'Student 316', age: 20 },
  { name: 'Student 346', age: 25 },
  { name: 'Student 930', age: 25 }
```

OTHER POINTS ABOUT LIMIT:

- The `limit` operator applies after filtering is complete. It retrieves the specified number of documents that match the filter criteria.
- You can combine `limit` with other operators like `sort` or projections for efficient and focused data retrieval.
- There's no minimum value for `limit`; you can set it to 0 to retrieve no documents. However, this might not be very useful in practice.
- MongoDB limits the maximum value for `limit` to be a 64-bit integer to prevent potential denial-of-service attacks by requesting an excessively large number of documents.

  The `limit` operator is a valuable tool for controlling the volume of data retrieved from your MongoDB collections. It enhances query performance, facilitates pagination, and optimizes resource utilization.

SELECTORS:

Selectors in MongoDB are essentially query criteria used to specify which documents to retrieve from a collection. They act like a filter, narrowing down the results based on specific conditions.

Understanding Selectors:

- Selectors are documents passed to the `find` method within MongoDB.
- These documents define the conditions that documents in the collection must meet to be included in the query results.
- Selectors can be simple or complex, involving various operators for filtering.

Basic Selector Example:

// Find all users with the username "john"

db.users.find({ username: "john" });

In this example, the selector is `{ username: "john" }`. This retrieves all documents in the "users" collection where the "username" field is equal to "john".

Common Selector Operators:

Comparison Operators:

- **`$eq`:** Matches documents where a field is equal to a specific value **(e.g. `{ age: { $eq: 30 } }`).**
- **`$gt`:** Matches documents where a field is greater than a value **(e.g., `{ price: { $gt: 100 } }`).**
- **`$lt`:** Matches documents where a field is less than a value **(e.g., `{ stock: { $lt: 5 } }`).**
- **`$ne`:** Matches documents where a field is not equal to a value **(e.g., `{ name: { $ne: "admin" } }`).**

Logical Operators:

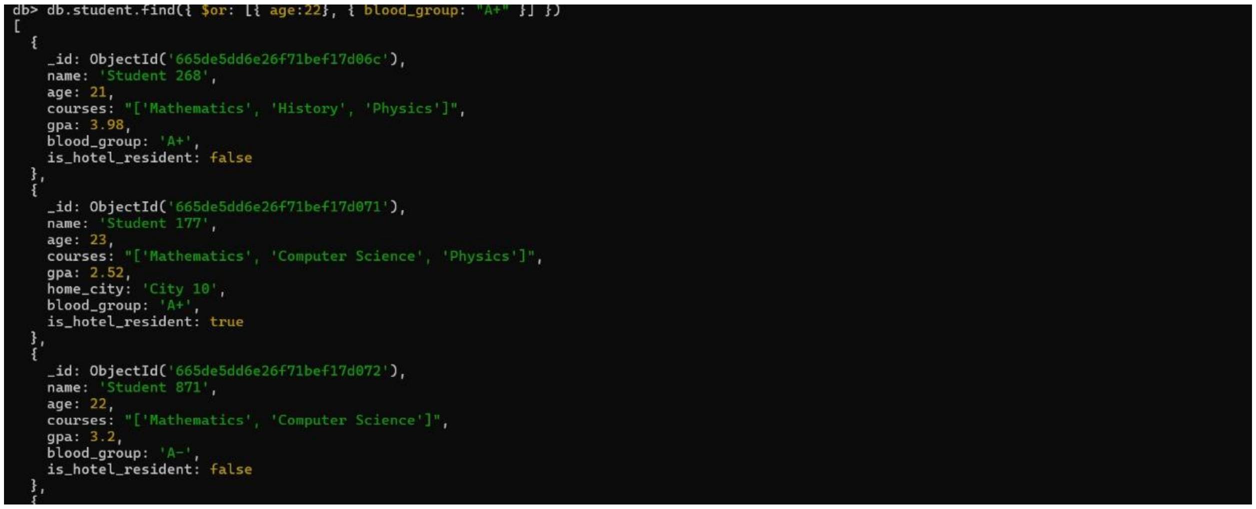
$and: Matches documents that meet all specified conditions (e.g., complex filtering).

EX: **db.products.find({ $and: [{ category: "electronics" }, { price: { $gt: 500 } }] })**

$or: Matches documents that meet at least one specified condition.

EX:

db.users.find({ $or: [{ username: "alice" }, { email: "bob@email.com" }] });



```
db> db.student.find({ $or: [{ age:22}, { blood_group: "A+" }] })
[
  {
    _id: ObjectId('665de5dd6e26f71bef17d06c'),
    name: 'Student 268',
    age: 21,
    courses: "['Mathematics', 'History', 'Physics']",
    gpa: 3.98,
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('665de5dd6e26f71bef17d071'),
    name: 'Student 177',
    age: 23,
    courses: "['Mathematics', 'Computer Science', 'Physics']",
    gpa: 2.52,
    home_city: 'City 10',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665de5dd6e26f71bef17d072'),
    name: 'Student 871',
    age: 22,
    courses: "['Mathematics', 'Computer Science']",
    gpa: 3.2,
    blood_group: 'A-',
    is_hotel_resident: false
  },
  {
```

$not: Inverts the match criteria (e.g., find inactive users).
EX: **db.users.find({ active: { $not: true } });**

Element Operators:

- **$in:** Matches documents where a field contains a value within a specified array (e.g., find products in specific categories).
  EX:
  db.products.find({ category: { $in: ["clothing", "accessories"] } });

  $nin: Matches documents where a field does not contain a value within a specified array.

Additional Selector Operations:

- **Regular Expressions:** Use $regex for pattern matching in text fields (e.g., search for usernames starting with "A").
- **Array Operators:** Filter documents based on elements within arrays (e.g., find orders with a specific item).

Combining Selectors with Operations:

- Selectors can be combined with other operations like:
  - **Projections:** Specify which fields to retrieve from documents.
  - **Limit:** Restrict the number of documents returned by a query.

    - **Skip:** Skip a specific number of documents before starting retrieval (useful for pagination

EXAMPLE:

```
// Find the first 3 active users sorted by name (ascending) and only
retrieve name and email fields
db.users.find({ active: true }, { projection: { name: 1, email: 1 },
limit: 3, sort: { name: 1 } });
```

Benefits of Using Selectors and Operators:

- **Precise Data Retrieval:** Target specific documents based on various criteria, ensuring you only fetch the data you need.
- **Enhanced Performance:** Focused queries using selectors and operators improve performance by reducing the amount of data scanned.
- **Flexible Data Manipulation:** Operators enable complex filtering, sorting, and data manipulation within your queries.