BENEFITS OF PROJECTION

1. **Reduced Data Transfer**:

By projecting only the necessary fields, you minimize the amount of data transferred over the network, which can significantly improve performance, especially when dealing with large documents or datasets.

2. **Improved Query Performance**:

Fetching only the required fields reduces the amount of data that MongoDB needs to process and retrieve from the disk, resulting in faster query execution.

3. **Lower Memory Usage**:

When you project only the needed fields, the result set is smaller, which can help reduce memory consumption on the client-side application.

4. **Increased Read Efficiency**:

By narrowing down the data fetched from the database, projections can help reduce the read load on the database, making it more efficient and scalable under high load conditions.

5. **Simpler Data Handling**:

Returning only the relevant fields makes the application code simpler and easier to manage, as it does not need to deal with unnecessary data.

6. **Security and Privacy**:

Projections can help enforce security and privacy by ensuring that sensitive fields are not exposed to unauthorized users or systems.

LIMIT AND SELECTORS

Limit:

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.

Syntax:

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

```
Type "it" for more
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: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
```

Limits two document:

```
db.gfg.find().limit(2)
```

Here, we only want the first two documents in the result. So, we pass 2 in the limit method.

Selectors:

- Comparison gt lt
- AND operator
- OR operator

Comparision gt lt:

The **$gt** operator allows us to find the documents where a specific field value is greater than an argument value. This operator can only accept numerical values. It can also be used with find and update operation functions like **find()**, **update()**, **updateMany()**, **etc**.

On the other hand, the **$lt** operator allows us to match documents whose specified field value is less than the argument value. The rest of the things are the same.

Note: The $gt and $lt operators can be clubbed with other operators. In the upcoming examples, we have used it with the $set operator to update values.

Syntax of $gt:

The **$gt** filters documents having a specific field value greater than the value passed as an argument. Following is the syntax of the **$gt** operator.

```
{field: {$gt: value}}
```

Here, the **field** is the field name where we are looking for the values that are greater than the argument **value**.

Syntax of $lt:

The **$lt** filters documents having a specific field value less than the value passed as an argument. Following is the syntax of the **$lt** operator.

```
{field: {$lt: value}}
```

Here, the **field** is the field name where we are looking for the values that are less than the argument **value**.

```
db> db.stud.find({age:{$gt: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
```

To find all students with age greater than 20.

AND Operator:

`AND` operation on an array of *one or more* expressions (`<expression1>`, `<expression2>`, and so on) and selects the documents that satisfy *all* the expressions.

`Syntax`:

```
{ $and: [ { <expression1> }, { <expression2> } , ... , { <expressionN> } ] }
```

Behavior

When evaluating the clauses in the `$and` expression, MongoDB's query optimizer considers which indexes are available that could help satisfy clauses of the `$and` expression when selecting the best plan to execute.

To allow the query engine to optimize queries, `$and` handles errors as follows:

- If any expression supplied to `$and` would cause an error when evaluated alone, the `$and` containing the expression may cause an error but an error is not guaranteed.
- An expression supplied after the first expression supplied to `$and` may cause an error even if the first expression evaluates to `false`.

For example, the following query *always* produces an error if `$x` is `0`:

```
db.example.find( {
$expr: { $eq: [ { $divide: [ 1, "$x" ] }, 3 ] }
} )
```

The following query, which contains multiple expressions supplied to `$and`, *may* produce an error if there is any document where `$x` is `0`:

```
db.example.find( {
$and: [
{ x: { $ne: 0 } },
{ $expr: { $eq: [ { $divide: [ 1, "$x" ] }, 3 ] } }
]
} )
```

```
db.stud.find({
$and:[
{home_city: "City 2"},
{blood_group: "B+" }
]
});


_id: ObjectId('665a89d776fc88153fffc0b4'),
name: 'Student 504',
age: 21,
courses: "['Physics', 'Computer Science', 'English', 'Mathematics']",
gpa: 2.42,
home_city: 'City 2',
blood_group: 'B+',
is_hotel_resident: true
,

_id: ObjectId('665a89d776fc88153fffc0eb'),
name: 'Student 367',
age: 19,
courses: "['English', 'Physics', 'History', 'Mathematics']",
gpa: 2.81,
home_city: 'City 2',
blood_group: 'B+',
is_hotel_resident: false
,
```

The above code helps to find the students from “city 2” with the blood group “B+”.

OR Operator:

MongoDB provides different types of logical query operators and $or operator is one of them. This operator is used to perform logical OR operation on the array of two or more expressions and select or retrieve only those documents that match at least one of the given expression in the array.

Syntax:

```
{ $or: [ { Expression1 }, { Expression2 }, ..., { ExpressionN } ] }
```

```
// 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 } }
  ]
});
```