

NoSQL



Agenda

1

Introduction

2

What, Why and When

3

SQL vs NoSQL

4

Installation of MongoDB

5

Create Databases

6

Create Collection

7

Create Documents

8

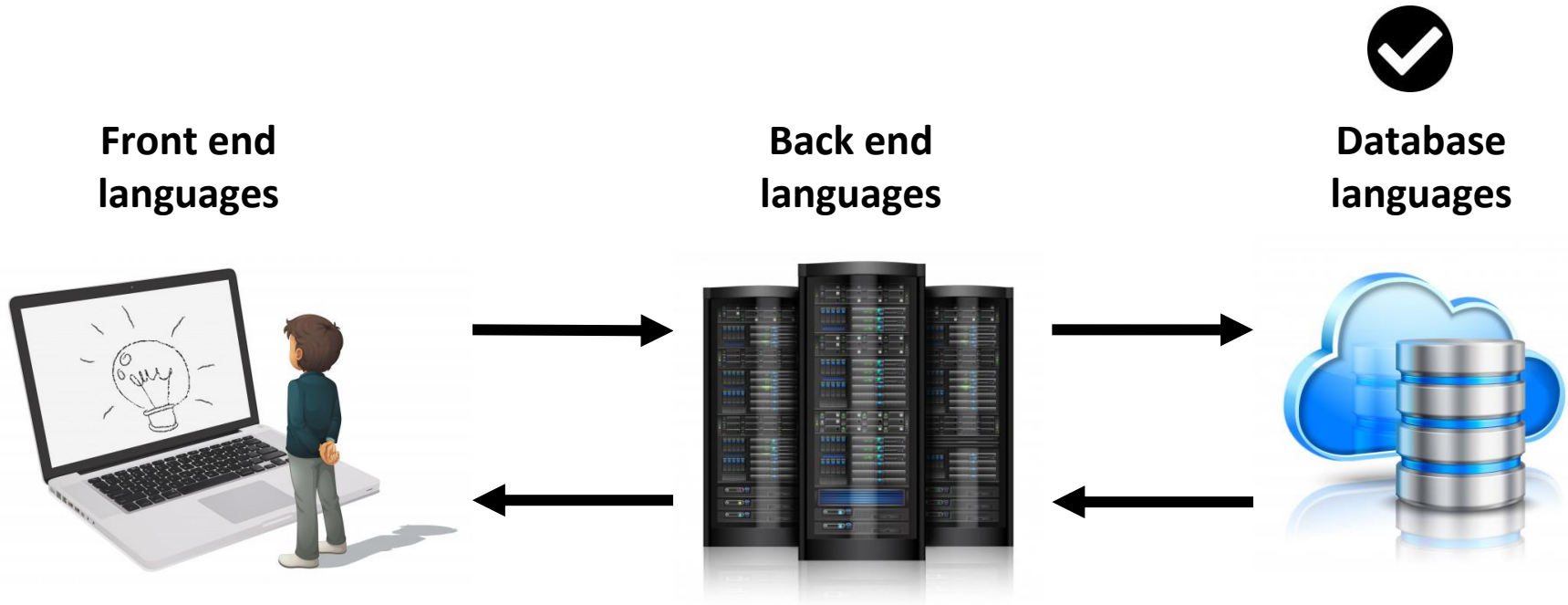
CRUD Operations

8

Query and Criteria

Introduction

Introduction



Introduction

A database Management System provides the mechanism to store and retrieve the data. There are different kinds of database Management Systems:

1. RDBMS (Relational Database Management Systems)
2. OLAP (Online Analytical Processing)
3. NoSQL (Not only SQL)

Introduction to NoSQL

Limitations of RDBMS

1. In relational database we need to define structure and schema of data first and then only we can process the data.
2. Relational database systems provides consistency and integrity of data by enforcing ACID properties (Atomicity, Consistency, Isolation and Durability). However in most of the other cases these properties are significant performance overhead and can make your database response very slow.
3. Most of the applications store their data in JSON format and RDBMS don't provide you a better way of performing operations such as create, insert, update, delete etc on this data.

What is NoSQL Database?

Not Only SQL (NoSQL) or non relational databases provides a mechanism for storage and retrieval of data other than tabular relations model used in relational databases. NoSQL database doesn't use tables for storing data. It is generally used to store big data and real-time web applications.

Why NoSQL Database?

NoSQL databases were created in response to the limitations of traditional relational database technology.

When compared against relational databases, NoSQL databases are more scalable and provide superior performance, and their data model addresses several shortcomings of the relational model.

The advantages of NoSQL include being able to handle:

- Large volumes of structured, semi-structured, and unstructured data
- Agile sprints, quick iteration, and frequent code pushes
- Object-oriented programming that is easy to use and flexible
- Efficient, scale-out architecture instead of expensive, monolithic architecture

When to use NoSQL Database?

- When you want to store and retrieve huge amount of data.
- The relationship between the data you store is not that important
- The data is not structured and changing over time
- Constraints and Joins support is not required at database level
- The data is growing continuously and you need to scale the database regular to handle the data.

Where to use NoSQL Database?

- Data isn't relational (e.g. Documents)
- Too much data to fit in a relational database

SQL vs NoSQL

Session Agenda

SQL vs NoSQL

- **Introduction** to SQL and NoSQL
- **Detailed comparison:** SQL vs NoSQL
- **The Final Conclusion**



Introduction to SQL and NoSQL

Introduction to SQL and NoSQL



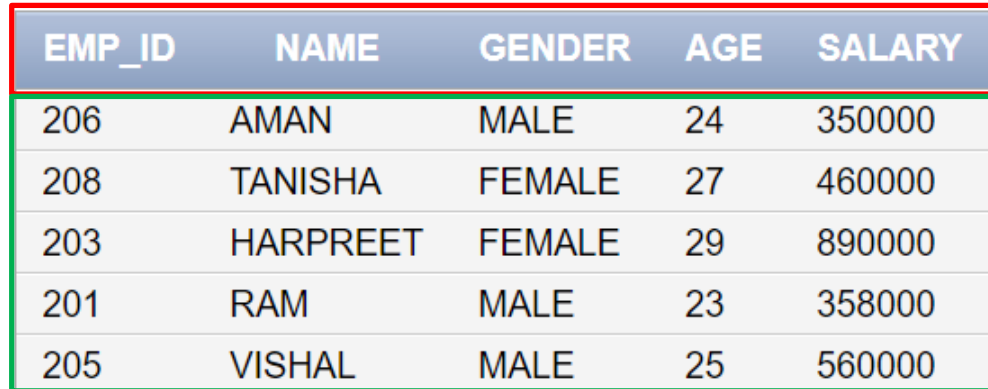
These two terms:

- Structured Query Language (**SQL**) is a query language for storing, manipulating and retrieving data stored in a Table structure (relational database).
- Not Only SQL (**NoSQL**) provides a mechanism for storage and retrieval of data other than tabular relations model used in relational databases (non relational database).

Structured Query Language (SQL)

Records (Rows)

Fields (Columns)



EMP_ID	NAME	GENDER	AGE	SALARY
206	AMAN	MALE	24	350000
208	TANISHA	FEMALE	27	460000
203	HARPREET	FEMALE	29	890000
201	RAM	MALE	23	358000
205	VISHAL	MALE	25	560000

EMPLOYEE (Table)

Not Only SQL (NoSQL)



SQL *vs* NoSQL

SQL vs NoSQL

1. Origin:



- **SQL** were developed in the 1970s with a focus on reducing data duplication as storage was much more costly than developer time.
- **NoSQL** were developed in the late 2000s with a focus on scaling, fast queries, allowing for frequent application changes, and making programming simpler for developers.

SQL vs NoSQL

2. Database:



- **SQL** databases are primarily called RDBMS or Relational Databases
- **NoSQL** databases are primarily called as Non-relational or distributed database

SQL vs NoSQL

3. Type:

SQL

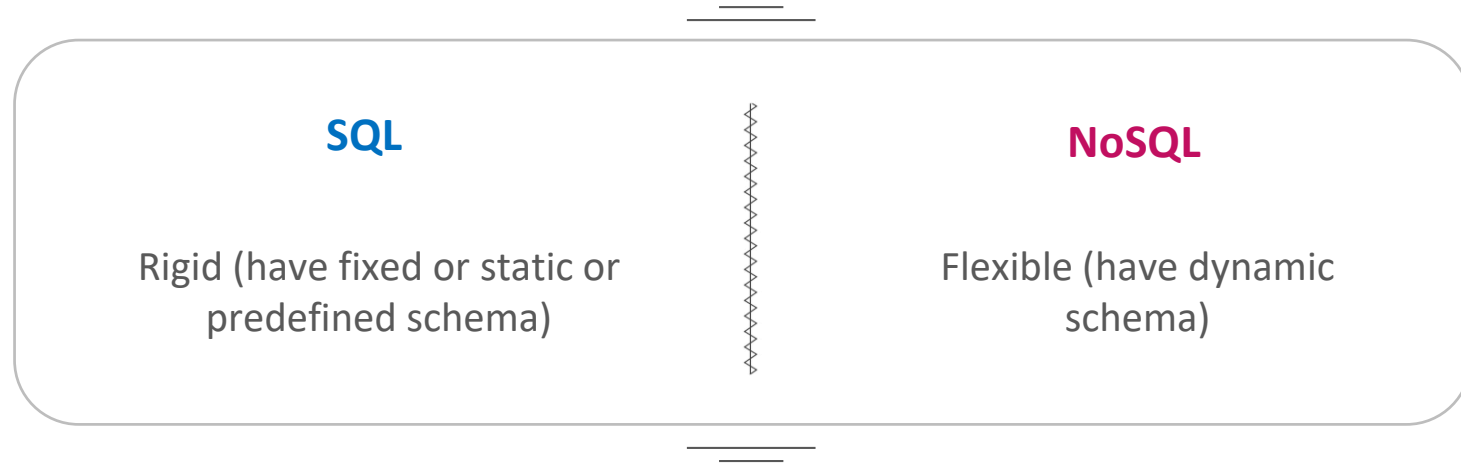
SQL databases are table based databases

NoSQL

NoSQL databases can be document based, key-value pairs or graph databases

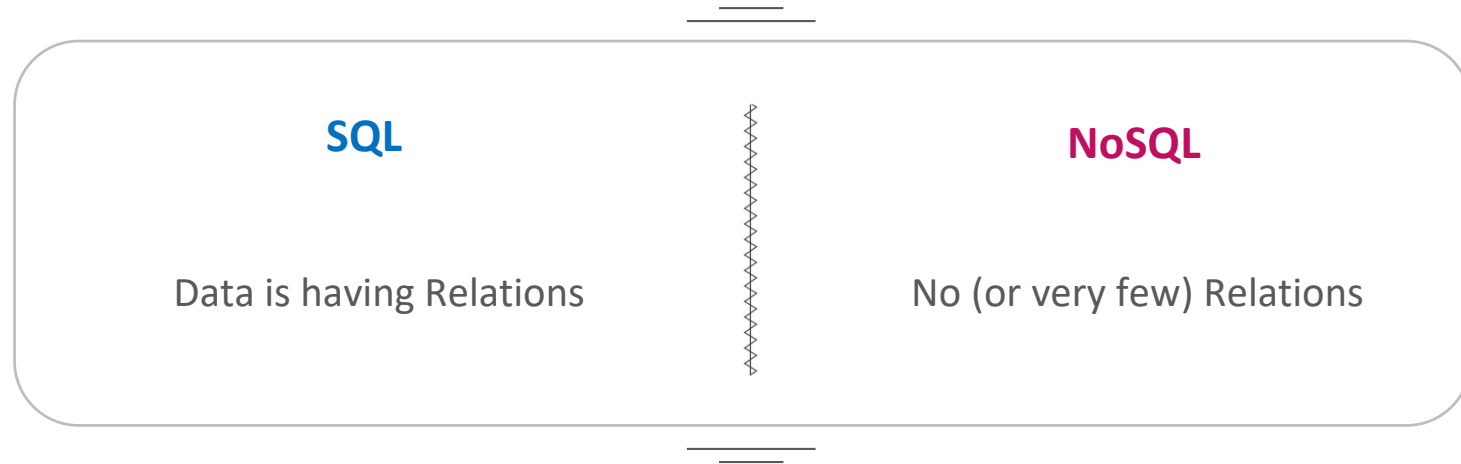
SQL vs NoSQL

4. Schemas:



SQL vs NoSQL

5. Relation:



SQL vs NoSQL

6. Distribution:

SQL

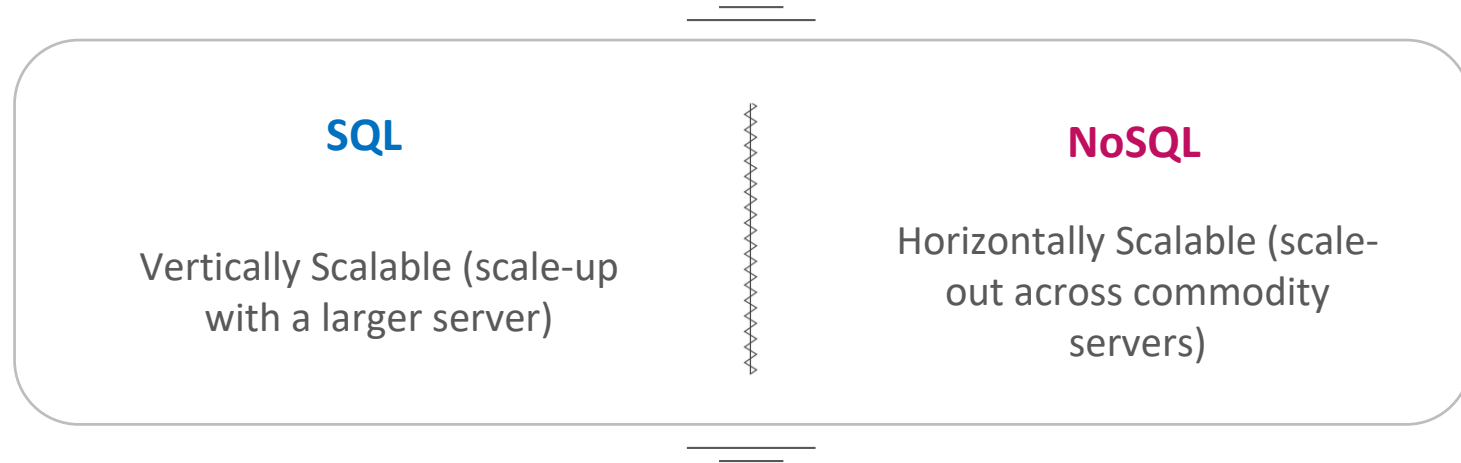
Data is distributed across multiple tables

NoSQL

Data is typically merged/nested in a few collections

SQL vs NoSQL

7. Scaling:



SQL vs NoSQL

8. Complex queries:

SQL

These are best suited for complex queries.

NoSQL

These are not so good for complex queries.

SQL vs NoSQL

9. Performance:

SQL

Requires specialized DB hardware for better performance.

NoSQL

Uses commodity hardware.

SQL vs NoSQL

10. Examples:

SQL

MySQL, Oracle, Sqlite,
PostgreSQL and MS-SQL etc.

NoSQL

MongoDB, BigTable, Redis,
RavenDB, Cassandra, Hbase,
Neo4j, CouchDB etc.

The Final Conclusion

Which approach you should go for?

There is no clear border, both approach is used for different cases

- **SQL** is used to carry operations on RDBMS, when creating custom dashboards, it is preferred while executing complex queries.
- **NoSQL** is used when the data needs a flexible schema and when constraints, validation logic are not essentially to be used in the databases.

Installation

Installation

- `C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe --version`
- `C:\Program Files\MongoDB\Server\4.4\bin\mongod.exe --version`
- `C:\Program Files\MongoDB\Server\4.4\bin\mongod.exe`
- `C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe`
- `mongo`

Create Database

Create Database

Once you are on MongoDB shell, Use the following command to create a database

use database_name;

if the database doesn't exist, above command creates a new database otherwise opens the existing one.

Eg: use greatlearning

```
>  
>  
> use greatlearning  
switched to db greatlearning  
>
```

Note:

In MongoDB, databases hold collections of documents

Delete Database

Use the following command to delete a database

db.dropDatabase()

```
> db.dropDatabase() :  
{ "ok" : 1 }  
>
```

List all Databases

- To list down all the databases, use the command below

show dbs

- This command lists down all the databases and their size on the disk.

```
> show dbs
admin    0.000GB
config  0.000GB
local    0.000GB
```

Note:

As you can see that the database “greatlearning” that we have created is not present in the list of all the databases. This is because a database is not created until you save a document in it

Create Collection

Collection

- MongoDB stores documents in collections. Collections are analogous to tables in relational databases
- A collection exists within a single database. Collections do not enforce a schema.
- A collection may store documents those which are not same in structure because its schema free database.

Create Collection – Method 1

- In MongoDB you need not to create collection before you insert document in it. With a single command you can insert a document in the collection and the MongoDB creates that collection on the fly.

Syntax: `db.collection_name.insert({key:value, key:value...})`

- Eg: To create a collection in the database iprimeddb, we use the following command.

`db.gla.insert({name:"CSS",source:"GLA", type:"Front end"});`

```
> db.greatlearning.insertOne({name:"HTML", source:"GreatLearningAcademy", type:"Front end", videos:5, active:true})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("6050421c8300d7d14ad1d46d")
}
>
```

```
> db.gla.insert({name:"CSS", source: "GLA", type: "Front end"})
WriteResult({ "nInserted" : 1 })
>
```

Create Collection – Method 2

We can also create collection before we actually insert data in it. This method provides you the options that you can set while creating a collection.

Syntax: `db.createCollection(name, options)`

- name is the collection name
- options is an optional field that we can use to specify certain parameters such as size, max number of documents etc. in the collection.

Eg: `db.createCollection("PGprograms");`

```
> db.createCollection("PGprograms");
{ "ok" : 1 }
>
```

```
> db
greatlearning
> show collections
PGprograms
gla
greatlearning
>
```


Deleting the Collection

To drop a collection , first connect to the database in which you want to delete collection and then type the following command to delete the collection:

```
db.collection_name.drop()
```

Note: Once you drop a collection all the documents and the indexes associated with them will also be dropped. To preserve the indexes we use remove() function that only removes the documents in the collection but doesn't remove the collection itself and the indexes created on it.

Eg: db.gla.drop();

```
> db.gla.drop()
true
> db.PGprograms.drop()
true
> show collections
greatlearning
>
```

Create Documents

Documents

MongoDB stores data records as BSON documents. BSON is a binary representation of JSON documents.

MongoDB documents are composed of field-and-value pairs and have the following structure:

```
{
  field1: value1,
  field2: value2,
  field3: value3,
  ...
  fieldN: valueN
}
```

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```

← field: value
← field: value
← field: value
← field: value

Collection and Documents

```
> show dbs
admin      0.000GB
config     0.000GB
greatlearning 0.000GB
local      0.000GB
> db
greatlearning
> show collections
greatlearning
> db.greatlearning.find()
{ "_id" : ObjectId("6050421c8300d7d14ad1d46d"), "name" : "HTML", "source" : "GreatLearningAcademy", "type" : "Front end", "videos" : 5, "active" : true }
>
```

```
> db.greatlearning.find().pretty()
{
  "_id" : ObjectId("6050421c8300d7d14ad1d46d"),
  "name" : "HTML",
  "source" : "GreatLearningAcademy",
  "type" : "Front end",
  "videos" : 5,
  "active" : true
}
>
```

CRUD

Query - A query is a request for data or information from a database table or combination of tables or collections in case of NoSQL database. This data may be generated as results returned by query or as pictorials, graphs or complex results, e.g., trend analyses from data-mining tools.

Create – `db.collection.insertOne({key:value, key:value})`
`db.collection.insertMany({key:value,key:value})`

Read - `db.collection.find(query,projection)`

Update – `db.collection_name.updateOne(<filter>,<update>)`
`db.collection_name.updateMany(<filter>,<update>)`

Delete – `db.collection_name.deleteOne(Deletion_criteria)`
`db.collection_name.deleteMany(Deletion_criteria)`

CRUD

- We can also add criteria to our queries so that we can fetch documents based on certain conditions.
- Instead of fetching all the documents from collection, we can fetch selected documents based on a criteria.

Operation	Syntax	Example
Equality	{<key>:<value>}	db.Employee.find({city:"Bangalore"}).pretty();
Less Than	{<key>:{\$lt:<value>}}	db.Employee.find({age:{\$lt:25}}).pretty();
Less Than Equals	{<key>:{\$lte:<value>}}	db.Employee.find({age:{\$lte:25}}).pretty();
Greater Than	{<key>:{\$gt:<value>}}	db.Employee.find({age:{\$gt:25}}).pretty();
Greater Than Equals	{<key>:{\$gte:<value>}}	db.Employee.find({age:{\$gte:25}}).pretty();
Not Equals	{<key>:{\$ne:<value>}}	db.Employee.find({age:{\$ne:25}}).pretty();

CRUD

C:\ Command Prompt - mongo

```
> db.Employee.find({city:"Bangalore"}).pretty();
```

```
{
  "_id" : 111,
  "fname" : "Raghu",
  "lname" : "KN",
  "age" : 34,
  "department" : "Transformation",
  "basicsalary" : "15000",
  "city" : "Bangalore"
}
{
  "_id" : 114,
  "fname" : "Vivan",
  "lname" : "Shirag",
  "age" : 25,
  "department" : "IT",
  "city" : "Bangalore"
}
{
  "_id" : 115,
  "fname" : "Sharath",
  "lname" : "Shetty",
  "age" : 25,
  "department" : "Facility",
  "city" : "Bangalore"
}
```

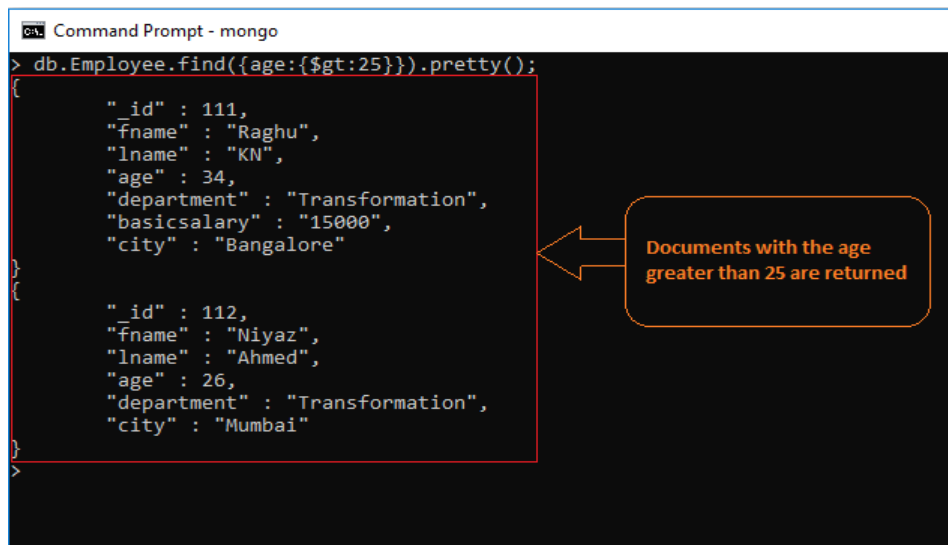
Document/s containing city as Bangalore is returned

"Greater than" Criteria

Used select documents which are greater than the specified criteria.

Syntax: `db.collections.find({fieldname:{$gt:criteria_value}}).pretty();`

Ex: `db.Employee.find({age:{$gt:25}}).pretty();`



```
Command Prompt - mongo
> db.Employee.find({age:{$gt:25}}).pretty();
{
  "_id" : 111,
  "fname" : "Raghu",
  "lname" : "KN",
  "age" : 34,
  "department" : "Transformation",
  "basicsalary" : "15000",
  "city" : "Bangalore"
}
{
  "_id" : 112,
  "fname" : "Niyaz",
  "lname" : "Ahmed",
  "age" : 26,
  "department" : "Transformation",
  "city" : "Mumbai"
}
>
```

Documents with the age greater than 25 are returned

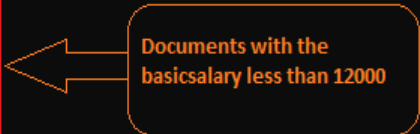
"Less than" criteria

Used select documents which are less than the specified criteria.

Syntax: `db.collections.find({fieldname:{$lt:criteria_value}}).pretty();`

Ex: `db.Employee.find({basicsalary:{$lt:12000}}).pretty();`

```
Command Prompt - mongo
> db.Employee.find({basicsalary:{$lt:12000}}).pretty();
{
  "_id" : 115,
  "fname" : "Sharath",
  "lname" : "Shetty",
  "age" : 25,
  "department" : "Facility",
  "basicsalary" : 10000,
  "city" : "Bangalore"
}
{
  "_id" : 116,
  "fname" : "Rithwik",
  "lname" : "Raj",
  "age" : 23,
  "department" : "HR",
  "basicsalary" : 10000,
  "city" : "Mysore"
}
```



"Not equal to" Criteria

```
Command Prompt - mongo
> db.Employee.find({city:{$ne:"Bangalore"}}).pretty();
{
  "_id" : 112,
  "fname" : "Niyaz",
  "lname" : "Ahmed",
  "age" : 26,
  "department" : "Transformation",
  "basicsalary" : 18000,
  "city" : "Mumbai"
}
{
  "_id" : 113,
  "fname" : "Subiya",
  "lname" : "Siraj",
  "age" : 4,
  "department" : "Transformation",
  "basicsalary" : 12000,
  "city" : "Mysore"
}
{
  "_id" : 116,
  "fname" : "Rithwik",
  "lname" : "Raj",
  "age" : 23,
  "department" : "HR",
  "basicsalary" : 10000,
  "city" : "Mysore"
}
>
```

Documents with the city other than Bangalore returned

"Greater than equals" Criteria

Command Prompt - mongo

```
> db.Employee.find({basicsalary:{$gte:12000}}).pretty();
```

```
{
  "_id" : 112,
  "fname" : "Niyaz",
  "lname" : "Ahmed",
  "age" : 26,
  "department" : "Transformation",
  "basicsalary" : 18000,
  "city" : "Mumbai"
}
{
  "_id" : 113,
  "fname" : "Subiya",
  "lname" : "Siraj",
  "age" : 4,
  "department" : "Transformation",
  "basicsalary" : 12000,
  "city" : "Mysore"
}
{
  "_id" : 114,
  "fname" : "Vivan",
  "lname" : "Shirag",
  "age" : 25,
  "department" : "IT",
  "basicsalary" : 20000,
  "city" : "Bangalore"
}
>
```

Documents with the basicsalary greater than equal to 12000 are returned

"AND" Criteria

Syntax:

```
db.mycollection.find(  
  {  
    $and:  
    [  
      {key1: value1}, {key2:value2}  
    ]  
  }  
).pretty()
```

Command Prompt - mongo

```
> db.Employee.find({$and:[{basicSalary:20000},{city:"Bangalore"}]}).pretty();
```

```
{  
  "_id" : 114,  
  "fname" : "Vivan",  
  "lname" : "Shirag",  
  "age" : 25,  
  "department" : "IT",  
  "basicSalary" : 20000,  
  "city" : "Bangalore"  
}
```

Documents with the
basicSalary 20000 and city
bangalore are returned

Ex:

```
db.Employee.find({$and:[{basicSalary:20000},{city:"Bangalore"}]}).pretty();
```

"OR" Criteria

Syntax:

db.mycollection.find

Ex:

db.Employee.find

```
> db.Employee.find({$or:[{basicSalary:20000},{city:"Bangalore"}]}).pretty();
{
  "_id" : 111,
  "fname" : "Raghu",
  "lname" : "KN",
  "age" : 34,
  "department" : "Transformation",
  "basicSalary" : "15000",
  "city" : "Bangalore"
}

{
  "_id" : 114,
  "fname" : "Vivan",
  "lname" : "Shirag",
  "age" : 25,
  "department" : "IT",
  "basicSalary" : 20000,
  "city" : "Bangalore"
}

{
  "_id" : 115,
  "fname" : "Sharath",
  "lname" : "Shetty",
  "age" : 25,
  "department" : "Facility",
  "basicSalary" : 10000,
  "city" : "Bangalore"
}
```

Documents with
basicSalary 20000 or
city Bangalore are
returned

"AND and OR together" Criteria

```
cmd Command Prompt - mongo
> db.Employee.find({basicsalary:{$gte:12000}, $or:[{city:"Bangalore"},{age:{$gte:22}}]}).pretty();
{
  "_id" : 111,
  "fname" : "Raghu",
  "lname" : "KN",
  "age" : 34,
  "department" : "Transformation",
  "basicsalary" : 15000,
  "city" : "Bangalore"
}
{
  "_id" : 112,
  "fname" : "Niyaz",
  "lname" : "Ahmed",
  "age" : 26,
  "department" : "Transformation",
  "basicsalary" : 18000,
  "city" : "Mumbai"
}
{
  "_id" : 114,
  "fname" : "Vivan",
  "lname" : "Shirag",
  "age" : 25,
  "department" : "IT",
  "basicsalary" : 20000,
  "city" : "Bangalore"
}
```

Documents with basicsalary greater than equal to 12000 and city as Bangalore or age greater than equal to 22 are returned



Thank You