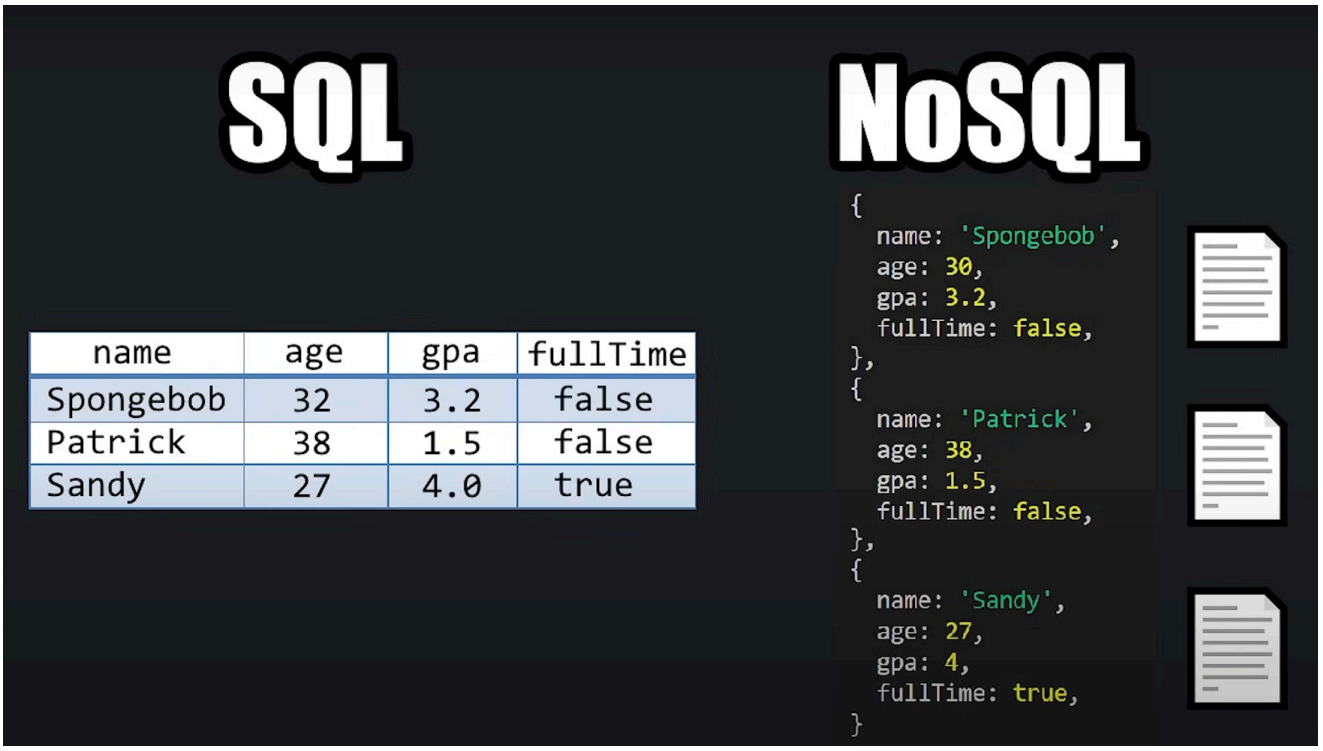
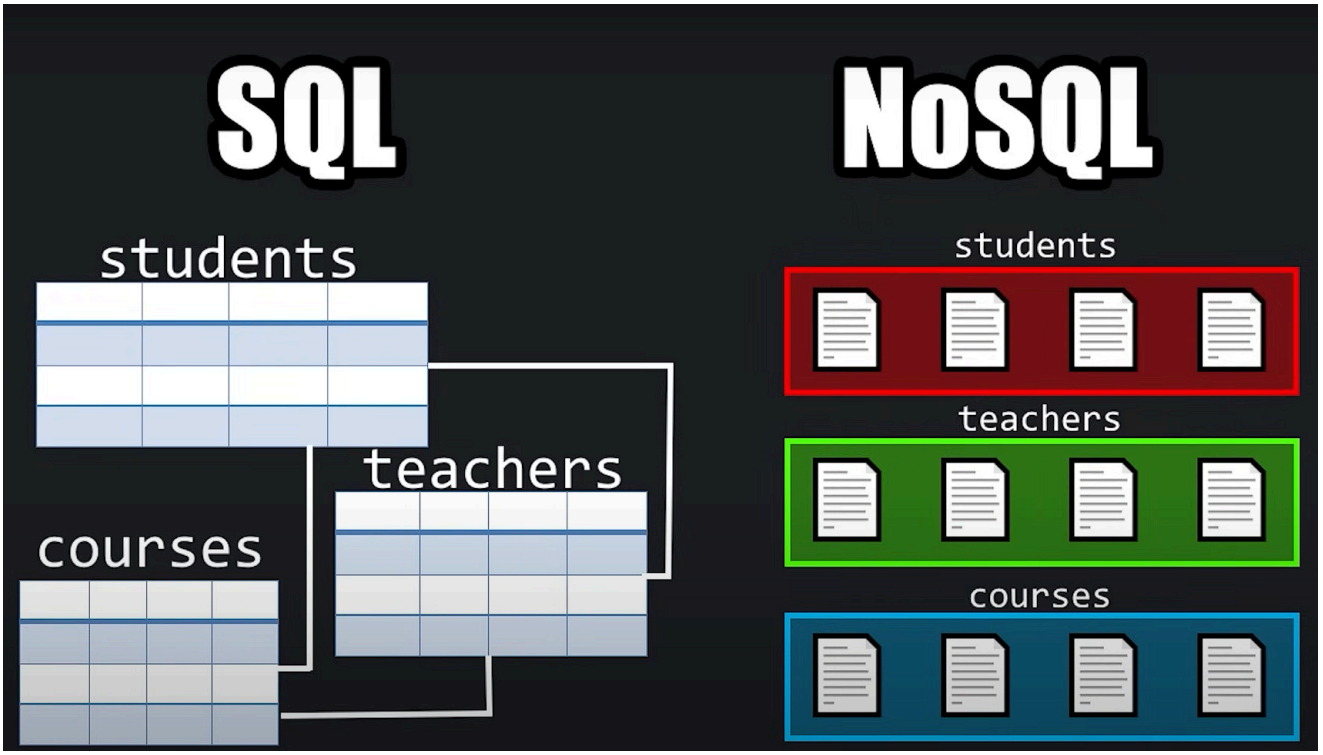
MongoDB is a **NoSQL database**—which means it stores data in a flexible, non-tabular format, unlike traditional SQL databases like MySQL or PostgreSQL.



The Data is stored in Document in MongoDB.

> `Document` --> Data in Document is stored as field value pair



```
Collections is a group of related documents :
 Example : Student, teachers, courses
```

> How Objects (data) is stored in Documents

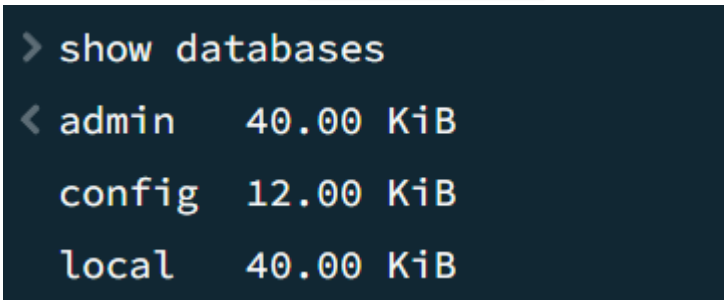
```
{
    name: 'yash'
    age: 20,
    cgpa:8.17
    fullTime: false,
}
```

`Installation Process`
Link : https://www.mongodb.com/try/download/community
Link : https://www.mongodb.com/try/download/shell

Commands for `Mongosh` or `MongoDB Compass`

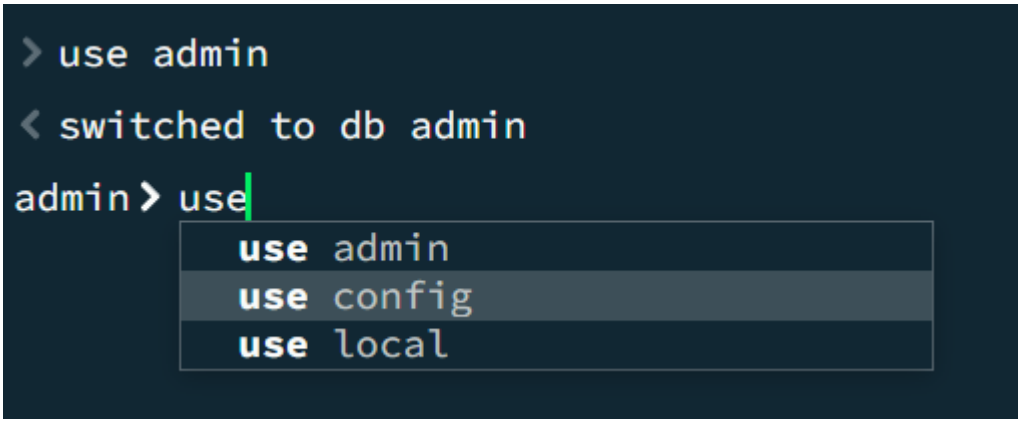
1. `Use` , `Create` , `Drop (Delete)` Database

1. Show Databases --> `show databases`

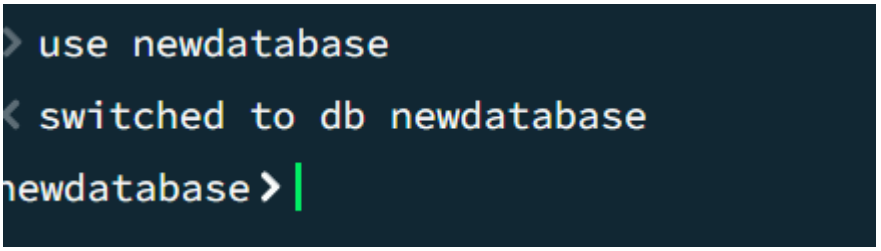


---

1. Select Database --> `use databasename`



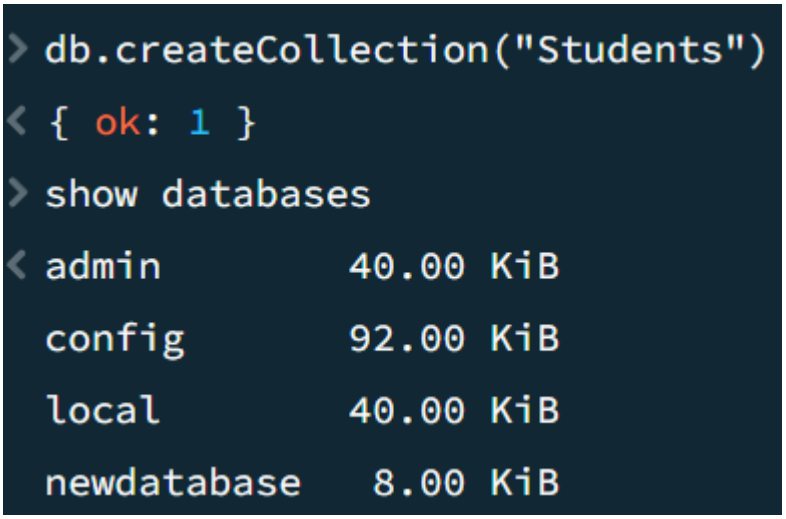
> we can create new database by `USE` command



---

3. `db.createCollection()` is a method in **MongoDB** used to **explicitly create a collection** in a database.

```
db.createCollection("Students")
```




```
db.createCollection("logs", {
  capped: true,
  size: 10485760,  // 10 MB
  max: 1000        // max 1000 documents
})
```

---

4. `db.dropDatabase()` is a **MongoDB command** used to **delete the current database** along with **all its collections and data**.

```
db.dropDatabase()
```

```
> show databases
< admin         40.00 KiB
  config        92.00 KiB
  local         40.00 KiB
  newdatabase    8.00 KiB
> use newdatabase
< already on db newdatabase
> db.dropDatabase()
< { ok: 1, dropped: 'newdatabase' }
> show databases
< admin   40.00 KiB
  config  92.00 KiB
  local   40.00 KiB
```

2. Operations on `Documents` & `Collections`

---

1. Adding Documents to the Collections

```
db.Students.insertOne({name:"Yash Pawar", age:22, percentage:80})

#we need to give collection name in Query like in above we have use
"Students"
```

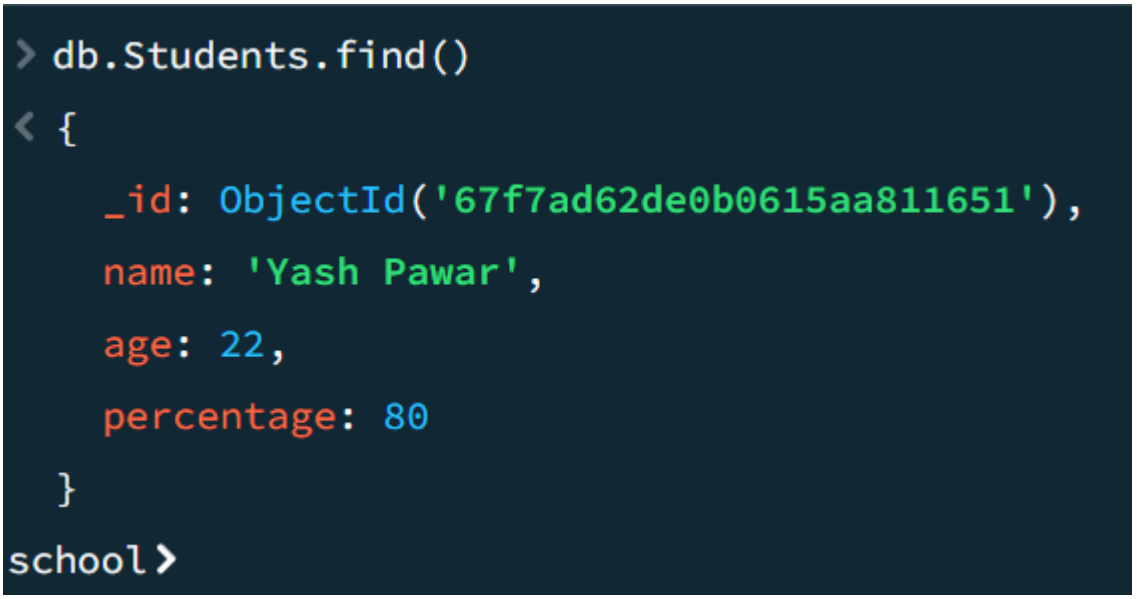
```
> db.Students.insertOne({name:"Yash Pawar", age:22, percentage:80})
< {
    acknowledged: true,
    insertedId: ObjectId('67f7ad62de0b0615aa811651')
  }
```

---

2. Find the Document in collection

```
db.Students.find()

#we need to give collection name in Query like in above we have use
"Students"
```



---

3. Insert `Many` Documents in Collection

`db.Students.insertMany( [{} , {} , {} ])`

```
> db.Students.insertMany([
    { name: "Harshal", age: 21, Percentage: 85 },
    { name: "Ayesha", age: 22, Percentage: 90 },
    { name: "Ravi", age: 20, Percentage: 78 },
    { name: "Neha", age: 23, Percentage: 88 }
  ])
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('67f7b1a97eab0627b2f9b740'),
      '1': ObjectId('67f7b1a97eab0627b2f9b741'),
      '2': ObjectId('67f7b1a97eab0627b2f9b742'),
      '3': ObjectId('67f7b1a97eab0627b2f9b743')
    }
  }
```

Find Documents `db.CollectionName.find()`

```
db.Students.find()
{
  _id: ObjectId('67f7ad62de0b0615aa811651'),
  name: 'Yash Pawar',
  age: 22,
  percentage: 80
}
{
  _id: ObjectId('67f7b1a97eab0627b2f9b740'),
  name: 'Harshal',
  age: 21,
  Percentage: 85
}
{
  _id: ObjectId('67f7b1a97eab0627b2f9b741'),
  name: 'Ayesha',
  age: 22,
  Percentage: 90
}
```

Datatypes in MongoDB

1. Type : `string`

- **Usage**: Most common data type. Used to store text data.

```
{ "name": "Alice" }
```

2. Type: `integer`

- `int32` or `int64` depending on the system or value.
- **Usage**: Used for whole numbers.

```
{ "age": 30 }
```

What's the difference between 32-bit and 64-bit integers?

Type	Syntax	Range	Use when...
`int32`	`NumberInt()`	-2,147,483,648 to 2,147,483,647	Number fits in regular range
`int64`	`NumberLong()`	-9,223,372,036,854,775,808 to +same	You need to store very large numbers

3. Type : `double`

- **Usage**: Stores floating point numbers.

```
{ "rating": 4.5 }
```

> NOTE :
> Used for precision but not as precise as `Decimal128`.
> Can lead to rounding issues in financial apps.

4. Type : `decimal`

- **Usage**: High-precision decimal type, useful for currency, finance, etc.

```
{ "balance": NumberDecimal("12345.67") }
```

5. Type : `bool`

- **Usage**: Stores `true` or `false`.

```
{ "isActive": true }
```

6. Type : `null`

- **Usage**: Represents an intentionally missing or empty value.

```
{ "middleName": null }
```

7. Type : `array`

- **Usage**: Stores multiple values in a single key.

```
{ "tags": ["mongodb", "nosql", "database"] }
```

8. **Embedded Document (Object)** `object`

- **Usage**: Stores nested documents (like JSON inside JSON).

```
{
  "profile": {
    "city": "Delhi",
    "zip": 110001
```

```
  }
}
```

9. Type : `objectId`

- **Usage**: A unique 12-byte ID for each document.

```
{ "_id": ObjectId("5f2b5e793b1d4f3c2c3e1f2a") }
```

```
db.Students.insertOne({
  name: "Suyash", //string
  age: 22,        //integer
  percentage: 78.9, //double
  fullTime: false,  //boolean
  registerDate: null, //null value
  gradutionDate: null,
  courses: ["Biology", "Chemistry", "Maths"], //array
  address: {                                  //embeded document
    street: "110 Samiksha vihar",
    city: "Pune",
    zip: 412207
  }
});
```

SORTING and LIMITING in MongoDB

`.sort()` – Sorting Documents

`order` :

- `1` → Ascending (smallest to largest, A to Z)
- `-1` → Descending (largest to smallest, Z to A)

```
--Sort students by percentage (highest first):

db.Students.find().sort({ percentage: -1 })
```

```
--Sort by name alphabetically

db.Students.find().sort({ name: 1 })
```

```
--sort multiple fields

db.Students.find().sort({ age: 1, name: -1 })
```

`.limit()` – Limit the Number of Results

- Limits the number of documents returned by the query.

```
--(number) : Maximum number of documents to return. eg.5

db.collection.find(query).limit(number)
```

```
db.Students.find().limit(5)
```

```
db.Students.find().sort({ percentage: -1 }).limit(3)
```

`.find()` - Finds documents or records

The `.find()` method is used to **search and retrieve** documents (records) from a MongoDB collection.

- Find all documents -

```
db.Students.find()
```

- Find with filters -

```
db.Students.find({ age: 22 })
```

- Find with Multiple conditions

```
db.Students.find({ age: 22, fullTime: true })
```

Find with Comparison Operators

MongoDB uses special operators like `$gt`, `$lt`, `$eq`, etc.

```
db.Students.find({ percentage: { $gt: 70 } })     --Greater than 70
db.Students.find({ age: { $gte: 20, $lte: 25 } }) --Between 20 and 25
```

✅ 1. `$gt` → Greater Than

Returns documents where the field's value is **greater than** the given value.
Finds students whose age is **greater than 20**.

```
db.Students.find({ age: { $gt: 20 } })
```

✅ 2. `$lt` → Less Than

Returns documents where the field's value is **less than** the given value.
Finds students with a percentage **less than 80**.

```
db.Students.find({ percentage: { $lt: 80 } })
```

✅ 3. `$gte` → Greater Than or Equal To

Finds students who are **18 or older**.

```
db.Students.find({ age: { $gte: 18 } })
```

✅ 4. `$lte` → Less Than or Equal To

Finds students with percentage **60 or below**.

```
db.Students.find({ percentage: { $lte: 60 } })
```

✅ 5. `$eq` → Equal To

Returns documents where the field's value **equals** the given value.

```
db.Students.find({ fullTime: { $eq: true } })
```

✅ 6. `$ne` → Not Equal

Finds students **whose age is not 22**.

```
db.Students.find({ age: { $ne: 22 } })
```

✅ 7. `$in` / `$nin` → Match any in array / exclude all in array

- Matches age = 21 OR 22 OR 23

```
db.Students.find({ age: { $in: [21, 22, 23] } })
```

- Excludes those in Pune or Mumbai

```
db.Students.find({ city: { $nin: ["Pune", "Mumbai"] } })`
```

`updateOne()` and `updateMany()` in MongoDB

Function	Updates...
`updateOne()`	Only the **first matching** document
`updateMany()`	**All documents** that match the filter

1. `updateOne()`

`SYNTAX` -->

```
db.collectionname.updateOne(
  { <filter> },
  { $set: { <field>: <new value> } }
)
```

`EXAMPLE` -->

```
db.Students.updateOne(
  { name: "Suyash" },
  { $set: { percentage: 90 } }
)
```

2. `updateMany()`

`SYNTAX -->`

```
db.collection.updateMany(
  { <filter> },
  { $set: { <field>: <new value> } }
)
```

`EXAMPLE -->`

```
db.Students.updateMany(
  { fullTime: false },
  { $set: { fullTime: true } }
)
```