🌟 Mongoose Operators

Mongoose lets you talk to your MongoDB database easily using JavaScript. But to **filter, search, and retrieve specific data**, we need to use **operators** — special keywords that help Mongoose understand *what exactly* we want from the database.

Think of them as **filters** or **conditions**, just like how you filter products on Amazon (price, brand, rating, etc.).

🧠 Why Use Operators?

Without operators, you'd only be able to do basic exact matching:

```js
Product.find({ price: 100 });
```

But what if you want:

- All products **above** ₹100?
- Posts with **likes more than 10** AND have an image?
- Products **not in** the "junk food" category?

That's where **operators** come in.

✅ Categories of Mongoose Operators:

1️⃣ Comparison Operators

Used when you want to **compare values** (like greater than, less than, not equal etc.)

Operator	Meaning	Example	What it does
`$eq`	Equal	`{ price: { $eq: 100 } }`	Same as `price == 100`
`$ne`	Not equal	`{ price: { $ne: 100 } }`	Price not equal to 100
`$gt`	Greater than	`{ price: { $gt: 100 } }`	Price > 100
`$gte`	Greater than or equal	`{ price: { $gte: 100 } }`	Price ≥ 100

`$lt`	Less than	`{ price: { $lt: 100 } }`	Price < 100
`$lte`	Less than or equal	`{ price: { $lte: 100 } }`	Price ≤ 100

🔍 **Example**: Find products that cost more than ₹500

```js
Product.find({ price: { $gt: 500 } });
```

---

2 Logical Operators

Used when you need to combine **multiple conditions** together (AND, OR, NOT, etc.)

Operator	Meaning	Example
`$and`	All conditions must be true	`{ $and: [{ inStock: true }, { price: { $lt: 100 } }] }`
`$or`	At least one condition must be true	`{ $or: [{ category: "fruit" }, { category: "veg" }] }`
`$not`	Inverts the result of a condition	`{ price: { $not: { $gt: 500 } } }`
`$nor`	None of the conditions must be true	`{ $nor: [{ inStock: false }, { price: { $gt: 500 } }] }`

🔍 **Example**: Products in stock AND less than ₹100

```js
Product.find({ $and: [ { inStock: true }, { price: { $lt: 100 } } ] });
```

🔍 **Example**: Products that are either fruits OR vegetables

```js
Product.find({ $or: [ { category: "fruit" }, { category: "veg" } ] });
```

---

3 Element Operators

Used to check if a field **exists** or check its **data type**

Operator	Meaning	Example

`$exists`	Checks if a field exists	`{ discount: { $exists: true } }`
`$type`	Checks the BSON type of a field	`{ quantity: { $type: "number" } }`

🔍 **Example**: Products with a `discount` field

```
js

Product.find({ discount: { $exists: true } });
```

🔍 **Example**: Products where `quantity` is a number

```
js

Product.find({ quantity: { $type: "number" } });
```

🛠️ TASK: Build a Product Management Express App to Practice Mongoose Operators

Objective:

Create an Express.js application that manages products and uses **Mongoose operators** for advanced querying. This will help you understand how to filter, search, and retrieve data from MongoDB efficiently.

Step 1: Setup Your Project

- Initialize a new Node.js app (`product-app`).
- Install dependencies: `express`, `mongoose`, `dotenv`, and `nodemon`.
- Connect your app to a MongoDB database using Mongoose and `.env`.

Step 2: Create a Product Model

Define a `Product` schema with the following fields:

- `name` (String)
- `price` (Number)
- `inStock` (Boolean)
- `category` (String)
- `discount` (Number)
- `quantity` (Number)

Step 3: Build API Endpoints

3.1 Add New Products

Create a **POST** route `/api/products` to add new products to the database.

3.2 Fetch Products with Filters

Create a **GET** route `/api/products/filter` that returns products based on various filters using Mongoose operators.

Step 4: Practice Using Mongoose Operators

In your filtering endpoint, implement queries that demonstrate these operator categories. Use query parameters or hardcoded examples.

1. Fetch products with `price` **greater than 100**.
2. Fetch products with `quantity` **less than or equal to 5**.
3. Fetch products where `price` is **not equal to 100**.
4. Fetch products that are **in stock AND have price less than ₹2d00**.
5. Fetch products that belong to category **"fruit" OR "veg"**.
6. Fetch products **not having price greater than 500**.
7. Fetch products where the `discount` field **exists**.
8. Fetch products where `quantity` is of type **number**.